Blockchain sin niebla

De la primera transacción
a tu servicio con RpcNode



En palabras sencillas · dibujos y ejemplos

En palabras sencillas

Blockchain sin niebla

De la primera transacción a tu propio servicio con RpcNode



Este libro es para ti si

- has oído la palabra «blockchain», pero no entiendes cómo funciona;
- quieres aceptar criptomonedas en tu sitio web;
- escribes programas y todavía nunca has consultado un nodo.

Qué vas a necesitar

- curiosidad: no hace falta saber nada de antemano;
- una computadora con terminal para repetir los ejemplos;
- las primeras consultas funcionan sin registro ni clave.

Cómo leer los capítulos



Autor: rpcnode.dev · 2026

Índice

Parte 1. Por qué hace falta un proveedor

1. Quiero aceptar cripto 4
2. Cuánto cuesta hacerlo uno mismo 9
3. Qué ofrece un proveedor 15

Parte 2. Blockchain desde cero

4. Dinero sin intermediario 21
5. Bloques y cadena 26
6. Claves, direcciones y firma 31
7. El viaje de una transacción 36
8. Cómo se pone de acuerdo la red 41
9. Monedas, tokens y contratos 47

Parte 3. Nodos y RPC

10. Qué tipos de nodos hay 53
11. Cómo se habla con un nodo 59
12. Un nodo propio por dentro 66
13. Cómo elegir proveedor 72

Parte 4. Cómo funcionan las redes

14. Bitcoin y sus parientes 77
15. Ethereum: cuentas, gas y eventos 82
16. La familia de Ethereum y las L2 88
17. TRON: energía en lugar de gas 93
18. Solana: todo en cuentas 99
19. TON: cartas en lugar de llamadas 105
20. Otros enfoques: de Sui a Cosmos 111
21. Qué red elegir 118

Parte 5. RpcNode

22. Primeros cinco minutos en RpcNode 125
23. Qué pasa con una solicitud 131
24. Historia y datos de archivo 137
25. Planes, precios y límites 143
26. Eventos: webhooks y sockets 149
27. Pagos y transacciones 155

Parte 6. Práctica

28. Montamos la tienda 161
29. Cuando todo sale mal 168
30. Seguridad antes del lanzamiento 175

Parte 7. Y ahora qué

31. DeFi: finanzas sin intermediario 182

Al final del libro

Conclusión 189
Apéndice A — Glosario 191
Apéndice B — Las redes en una página 193
Apéndice C — Guía rápida de consultas 195

Capítulo 1. Quiero aceptar cripto: ¿qué necesito para eso?

Todo empezó con un cliente

Artyom tiene una pequeña tienda en línea: tuesta café y lo envía por correo. Tarjetas, transferencias bancarias, pago contra entrega: todo funciona ya. Y un día llega este mensaje:

> «¡Hola! Quiero pedir dos kilos. ¿Puedo pagar con criptomonedas?»

A Artyom le parece bien. Instala una billetera en el teléfono, obtiene una dirección y se la envía al cliente. A los pocos minutos, el dinero aparece en la pantalla.

Al décimo pedido, revisar la billetera a mano ya cansa. Quiere que el sitio sepa por sí solo que el pedido está pagado.



Entendamos: qué es un nodo

Para averiguar algo sobre una blockchain, hay que preguntarle a quien la guarda. Ese alguien se llama **nodo**.

Imagina un cuaderno común donde está anotado quién le transfirió cuánto a quién. El cuaderno no está guardado en una única caja fuerte. Muchísimos participantes de todo el mundo tienen una copia y las comparan entre sí sin parar. Eso es la blockchain, y la veremos en detalle en el capítulo 5.

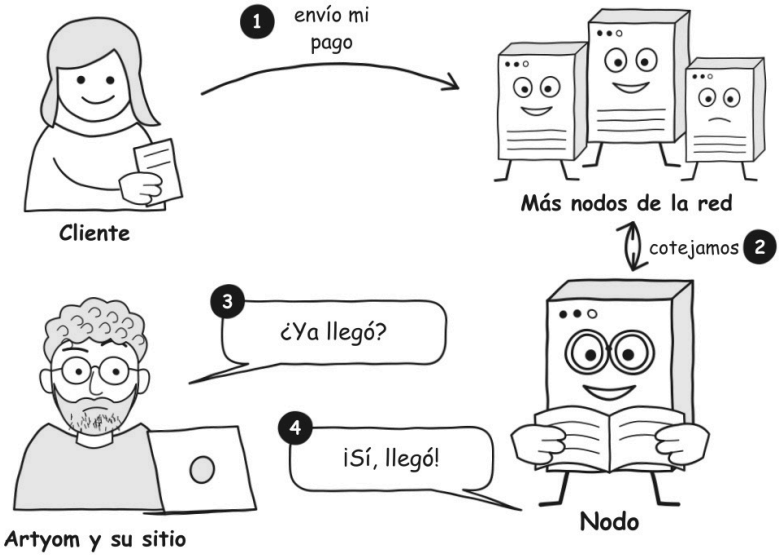
Un nodo es una computadora con el programa de la red. Hace cuatro cosas:

1. Guarda su propia copia de los registros.
2. Recibe registros nuevos de otros nodos y los comprueba según las reglas de la red.

3. Reenvía las transferencias que le mandan.
4. Responde a las preguntas de los programas: «¿cuál es el saldo?», «¿pasó la transferencia?».

Hay distintos tipos de nodos: algunos no guardan toda la historia. De eso hablamos en el capítulo 10.

El sitio de Artyom no mira dentro de la blockchain por su cuenta. Le hace una pregunta al nodo y el nodo responde.



Probemos: la primera pregunta a un nodo

Los programas hablan con el nodo siguiendo reglas sencillas: envían una solicitud y reciben una respuesta. Una de esas reglas se llama **JSON-RPC**. Esta es una pregunta a un nodo de la red Ethereum: «¿cuántas monedas hay en esta dirección?»

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

Aquí `URL_DEL_NODO` es a dónde llamamos, y `DIRECCION_DE_BILLETERA` es la dirección cuyo saldo preguntamos. La palabra `latest` significa «en el momento más reciente».

El nodo responde:

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

Parece un revoltijo de caracteres, pero es un número escrito en hexadecimal: 1 000 000 000 000 000 000. El nodo cuenta en las fracciones más pequeñas de la moneda, que se llaman **wei**. En un ETH hay exactamente esa cantidad. Es decir, en la dirección hay 1 ETH.

Así que todo el truco consiste en preguntar y responder. Solo queda decidir de quién será ese `URL_DEL_NODO`.

Dos caminos: nodo propio o proveedor

Puedes poner en marcha un nodo tú mismo. O puedes conectarte a uno ajeno, que alguien mantiene por ti. Ese servicio se llama **proveedor**. Te entrega un `URL_DEL_NODO` listo para usar. RpcNode, del que trata este libro, es uno de esos servicios.

Pregunta	Nodo propio	Proveedor
Cuándo se puede empezar	Tras instalar y sincronizar: de horas a semanas, según la red	Tras registrarte, en pocos minutos
Qué se paga	Servidor, discos, tráfico y el tiempo de un especialista	El plan del servicio
Quién actualiza y repara	Tú	El proveedor
Varias redes	Un nodo para cada una	Normalmente, desde un solo panel
Qué arriesgas	Que el nodo se atrase o se rompa mientras duermes	Dependes de un servicio ajeno

Los dos caminos tienen su lugar. Un nodo propio se justifica si necesitas una privacidad especial, si la carga es muy grande o si participas en el funcionamiento de la red.

> **Artyom:** «Yo tuesto café, no hago guardias junto a servidores. Pero, aun así, ¿cuánto cuesta hacerlo uno mismo?»

La respuesta corta: para varias redes, con servidores de reserva y la historia completa, la cuenta llega a decenas de miles de dólares. Un solo nodo pequeño puede salir mucho más barato. El cálculo detallado está en el capítulo 2.

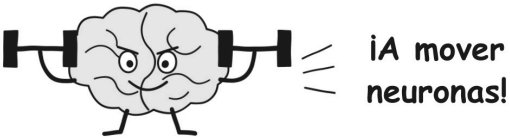
La trampa: «le doy al proveedor la clave de mi billetera»

Nunca. Para conocer un saldo o ver un pago, al nodo le basta con la dirección. La clave privada y la frase secreta mandan sobre el dinero: quien las conoce puede llevárselo todo. Un proveedor honrado no las necesita. Las claves las veremos en el capítulo 6.



Y una segunda advertencia. La consulta de arriba muestra solo la moneda propia de la red, aquí ETH. Si el cliente paga con un token, por ejemplo USDT, esa pregunta no lo verá. Para los tokens se pregunta de otra manera; de eso trata el capítulo 9.

Para pensar



1. Artyom comprueba un pago, pero su nodo lleva una hora de atraso respecto a la red. ¿Qué verá y qué le escribirá al cliente?
2. La tienda tiene tres pedidos al día. ¿Y si fueran tres mil? ¿Qué cambia en la elección entre nodo propio y proveedor?

Pista para la primera: un nodo atrasado muestra con toda honestidad lo que había hace una hora.

Lo esencial

- Un programa se entera de lo que pasa en la blockchain haciendo preguntas a un nodo.
- El nodo lo puedes mantener tú o contratarlo a un proveedor.
- El nodo necesita la dirección, no tu clave privada.

A continuación: capítulo 2. Artyom saca la calculadora y calcula cuánto le costarían sus propios servidores.

Capítulo 2. ¿Cuánto cuesta hacerlo todo uno mismo?

Artyom saca la calculadora

Artyom razona así de simple: «Compro un servidor, instalo un nodo y los pagos funcionarán solos. Pago una sola vez». Abre la calculadora y empieza a apuntar.



Entendamos: de qué se compone la cuenta

El servidor es solo la punta. Lo demás no se ve al principio.



1. **Servidor.** Procesador, memoria y discos NVMe rápidos.
2. **Servidor de reserva.** Si uno se rompe, el otro responde mientras reparan el primero.
3. **Los discos se llenan.** La blockchain solo escribe cosas nuevas. En un año, hasta noviembre de 2026, la cadena de Bitcoin creció de unos 700 a 786 GB.
4. **Tráfico.** El nodo intercambia datos sin parar. Un liteserver de TON necesita en las horas pico unos 16 TB al mes.
5. **Actualizaciones.** La red cambia sus reglas y el programa antiguo ya no da abasto.
6. **Personas.** Alguien tiene que configurar todo esto, vigilarlo y despertarse de noche.

Cuánto hardware pide cada red

Son los requisitos de un nodo normal según la documentación oficial, no para participar en las votaciones de la red.

Red	Procesador	Memoria	Disco
Bitcoin	desde 1 GHz	2 GB	recomiendan 750 GB, aunque la propia cadena ya ronda los 786 GB
Ethereum	4+ núcleos	32 GB	NVMe de 4 TB
BNB Smart Chain	16 núcleos	64 GB	SSD desde 3 TB; la instantánea completa de la base ya ronda los 7,4 TB, y la recortada, 1,9 TB
TRON	16 núcleos	32 GB	SSD desde 3,5–4 TB
Solana (nodo RPC)	16 núcleos o más	desde 512 GB	varios NVMe: cuentas 1 TB+, registro 1 TB+, instantáneas 500 GB+

¿Notas cosas raras? Para Bitcoin aconsejan un disco más pequeño que la propia cadena. Para BNB el mínimo son 3 TB, y la instantánea completa es más del doble. La documentación no siempre sigue el ritmo de la red, así que el disco se toma con margen.

¿Y si se necesita toda la historia?

Un nodo normal conoce bien el presente. Para preguntar «¿qué saldo había hace un año?» hace falta un nodo **de archivo**, que veremos en el capítulo 10. Es bastante más pesado. El archivo de TON necesita 16–20 TB. En Ethereum el tamaño depende mucho del programa: de 2 a 14 TB y más. Los archivos no entran en nuestro cálculo: con ellos saldría más caro.

Contamos el dinero

Un servidor se puede comprar, pero nosotros calcularemos el alquiler. Es una estimación basada en las tarifas públicas de empresas de hosting a 27 de noviembre de 2026, sin impuestos ni cargo de alta. Los precios cambian.

Clase de servidor	Qué incluye	Aprox. al mes
Mediano	8 núcleos, 64 GB, 2×2 TB NVMe	$180
Grande	16 núcleos, 128 GB, 4–8 TB NVMe	$460–560
Muy grande	512 GB de memoria o más	$1 160–2 350

Ahora, cinco redes. Tomamos precios cercanos al límite inferior y ponemos dos servidores por red: el principal y el de reserva.

Red	Clase	Un servidor, al mes	Con reserva, al mes
Bitcoin	mediano	$180	$360
Ethereum	mediano	$180	$360
BNB Smart Chain	grande	$500	$1 000
TRON	grande	$500	$1 000
Solana	muy grande	$1 200	$2 400
Total		**$2 560**	**$5 120**

Artyom: «5 120 dólares al mes son 61 440 al año. ¡¿Y eso es solo el alquiler?!»

Sí. Con los precios altos de la primera tabla saldrían unos $92 000 al año. De ahí lo de las «decenas de miles de dólares».

Mantener una sola red es mucho más barato. Bitcoin es poco exigente: le basta incluso un servidor más modesto que el mediano.

¿Y quién se va a ocupar de esto?

Los servidores no se mantienen solos. Según Habr Career, un ingeniero DevOps de nivel intermedio en Rusia gana aproximadamente entre 160 y 290 mil rublos al mes. Una sola persona no puede estar de guardia las 24 horas, sin fines de semana ni vacaciones. Incluso dos ya suman unos 3,8–7 millones de rublos al año.

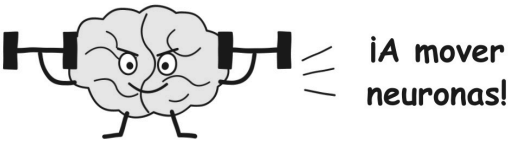
La trampa: «compro un servidor y listo»

Un solo servidor es un único punto de fallo. El disco se llenó, el programa se colgó, la red se actualizó, y el sitio ya no ve el pago. El cliente envió el dinero, pero el pedido figura como «no pagado».



Y algo más que se suele olvidar. Lee las normas del hosting antes de pagar: algunos prohíben los programas relacionados con criptomonedas y pueden apagarte el servidor.

Para pensar



1. Artyom solo necesita Bitcoin, con un servidor de reserva de clase mediana. ¿Cuánto es al año?
2. A las cinco redes se suma una sexta en servidores grandes de $500, también con reserva. ¿Cuánto sube la cuenta anual?
3. Los clientes pagan tres veces al día. ¿Justifican semejante gasto?

Respuestas: 1) 360 × 12 = $4 320. 2) 1 000 × 12 = $12 000. 3) Tú decides, y el capítulo 3 te ayudará.

Lo esencial

- No se paga una sola vez, sino cada mes.
- Cinco redes con reserva son más de $60 000 al año solo de alquiler, sin archivos ni personal.
- Los discos se llenan, las redes se actualizan y alguien tiene que estar de guardia.

A continuación: capítulo 3. De qué se encarga un proveedor y cuándo sí hace falta un nodo propio.

De dónde salen las cifras

Todas las páginas se consultaron el 27 de noviembre de 2026.

- Bitcoin, requisitos: bitcoin.org/en/bitcoin-core/features/requirements
- Bitcoin, tamaño de la cadena: 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
- Sueldos: career.habr.com/professions/devops (datos de 2025–2026)

Los precios de los servidores son una estimación redondeada del autor a partir de las tarifas públicas de varias empresas de hosting.

Capítulo 3. Qué ofrece un proveedor y cuándo sí hace falta un nodo propio

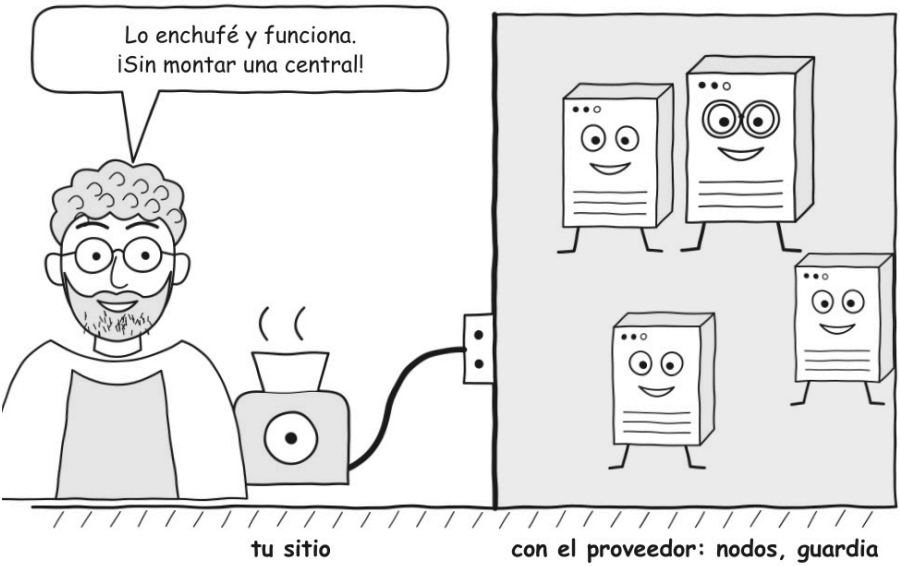
Artyom ante el cruce de caminos

Después de los cálculos del capítulo 2, la calculadora está boca abajo. Cinco redes con reserva son más de 60 000 dólares al año solo de alquiler. El café no da para tanto.



Entendamos: de qué se encarga un proveedor

La electricidad la tomas del enchufe; no construyes una central para usar la tetera. Con los nodos pasa igual: el **proveedor** los mantiene en su casa y a ti te da una dirección a la que preguntar.



Normalmente, de su lado quedan cinco tareas:

1. **Hardware y discos.** Comprar, alquilar y ampliar cuando se acaba el espacio.
2. **Sincronización.** El nodo no debe quedarse atrás respecto a la red.
3. **Actualizaciones.** La red cambia sus reglas y el programa se cambia a tiempo.
4. **Reemplazo de lo roto.** Las solicitudes se envían a un nodo sano.
5. **Guardias.** De noche no es Artyom quien se despierta.

Los proveedores son distintos y prometen cosas distintas. Qué recibes exactamente de esta lista se comprueba antes de pagar. Cómo elegir lo veremos en el capítulo 13.

Qué sigue siendo cosa tuya

- **Las claves de la billetera.** Solo las tienes tú.
- **La URL del endpoint.** Así se llama la dirección que te da el proveedor. A menudo lleva incrustada tu clave personal de acceso. Cuídala como una contraseña: las solicitudes ajenas gastarán tu límite.
- **Tu código.** Debe sobrevivir a un error, repetir la solicitud y no contar dos veces el mismo pago.

Tres palabras de cualquier plan

Palabra	Qué significa	Pregúntate
Solicitudes por segundo (RPS)	Con qué frecuencia se puede preguntar	¿Cuántos clientes llegarán a la vez?
Límite, o créditos	Cuánto se puede preguntar en total durante un periodo. Cada pregunta cuesta distinto	¿Me alcanzará hasta fin de mes?
Acceso de archivo	Si se puede preguntar por el pasado lejano	¿Necesito la historia antigua?

Probemos: una pregunta a un nodo de verdad

En el capítulo 1, en lugar de la dirección aparecía `URL_DEL_NODO`. Ahora pondremos una de verdad. RpcNode tiene nodos públicos gratuitos que no piden registro ni clave. Preguntemos el número del último bloque de la red Ethereum:

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

Esto contestó el nodo cuando se escribía este capítulo:

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

`0x1946393` es 26 502 035. Tu número será mayor: la red no se queda quieta. Repite la consulta al cabo de un minuto y lo verás.

Los nodos públicos son compartidos por todos y por eso tienen restricciones: a 27 de noviembre de 2026 son 5 solicitudes por segundo desde una misma dirección, y los métodos más «caros» están cerrados. Para conocerse basta. Una tienda necesita su propio endpoint.

Cuánto cuesta

Como ejemplo, los dos primeros planes de RpcNode a 27 de noviembre de 2026. Consulta los vigentes en el sitio.

	Free	Starter
Precio al mes	$0	$19
Créditos	15 millones	50 millones
Solicitudes por segundo	15	40
Endpoints	3	5
Redes	Ethereum y Solana	todas
Acceso de archivo	no	sí

Los nodos de RpcNode están en tres geografías: Fráncfort, Estados Unidos y Japón.

> **Artyom**: «¿19 dólares frente a 5 120 al mes? ¿Dónde está el truco?»

Son cosas distintas. Un servidor propio es todo tuyo. Con un plan recibes una parte de una capacidad compartida, con límites. A una tienda pequeña le basta esa parte y no tiene sentido pagar por una «central eléctrica» entera.

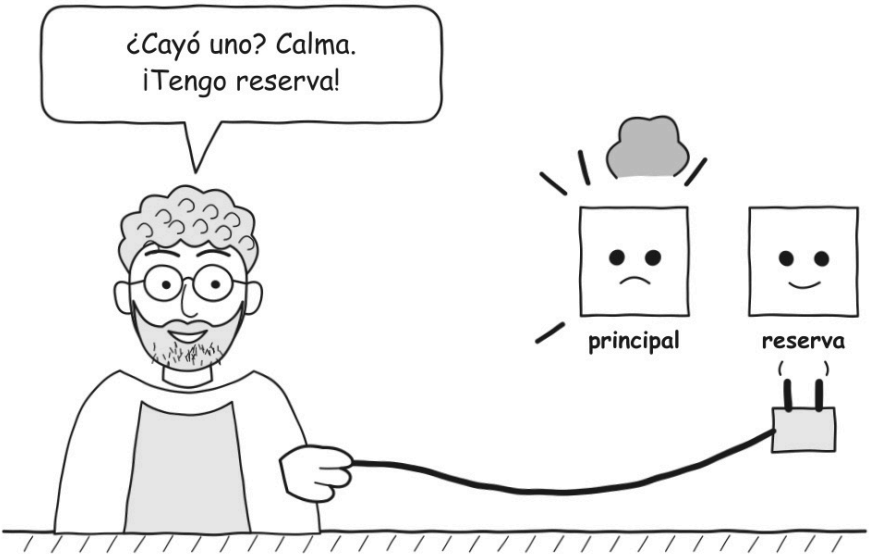
Cuándo sí hace falta un nodo propio

- **Privacidad.** El proveedor ve tus solicitudes: qué direcciones compruebas y desde dónde preguntas.
- **Carga enorme.** Hay tantas solicitudes que el plan sale más caro que tener servidores propios.
- **Participación en la red.** Eres validador o desarrollas la propia red.
- **Ajustes especiales.** Necesitas un método o un modo que el proveedor no ofrece.
- **Exigencias de la ley o del cliente.** Los datos deben procesarse solo en tu casa.

También existe un camino mixto: nodo propio para la red principal y proveedor para las demás y como respaldo.

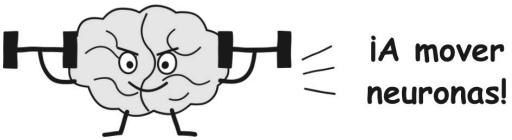
La trampa: «el proveedor nunca se va a caer»

Todo se cae: los servidores, las líneas de comunicación y también los servicios de los proveedores. Si el pago es importante para ti, decide de antemano qué hace el sitio cuando no hay respuesta.



- No le digas al cliente «no pagado» cuando simplemente no pudiste preguntar.
- Repite la solicitud más tarde, no cien veces seguidas.
- Ten una dirección de reserva: un segundo proveedor o un nodo propio.

Para pensar



1. El sitio de Artyom comprueba el pago una vez por segundo para cada uno de los 20 pedidos pendientes, con una solicitud por comprobación. ¿Cabe eso en 15 solicitudes por segundo?
2. ¿Qué debe ver el cliente si el nodo no responde durante diez minutos?
3. ¿Quién necesita un nodo propio: una tienda de café o un equipo que desarrolla la propia red?

Pista para la primera: 20 es más que 15. Comprueba con menos frecuencia o pregunta con más cabeza; cómo hacerlo, en el capítulo 26.

Lo esencial

- El proveedor se encarga del hardware, la sincronización, las actualizaciones y las guardias.
- Las claves, la URL del endpoint y el comportamiento del sitio ante los fallos siguen siendo cosa tuya.
- Un nodo propio hace falta por privacidad, carga grande, participación en la red o exigencias especiales.

Fin de la primera parte. Artyom eligió un proveedor y recibió respuesta de un nodo de verdad. Pero ¿qué es ese «bloque» número 26 502 035 que mencionó? **A continuación:** parte 2, capítulo 4. Dinero y registros sin intermediario.

Capítulo 4. Dinero y registros sin intermediario

¿Dónde está tu dinero?

Artyom recibió una transferencia en su tarjeta. Ningún billete viajó a ninguna parte. Entonces, ¿qué pasó? Alguien redujo el número junto al nombre del cliente y aumentó el número junto al nombre de Artyom.

El dinero bancario es una anotación. Y quien lleva las anotaciones se llama intermediario: un banco o un servicio de pagos.



Entendamos: para qué sirve el intermediario y en qué incomoda

El intermediario hace muchas cosas útiles. Comprueba que alcance el dinero. No deja gastarlo dos veces. Puede resolver una disputa y devolver un pago.

Pero todos los registros están en sus manos. Puede detener una transferencia, cobrar una comisión, bloquear una cuenta o estar cerrado en un día festivo. Se nota sobre todo cuando el cliente vive en otro país.

> **Artyom:** «Pues que el cliente me mande el dinero como un archivo. Como una foto. ¡Sin ningún intermediario!»

Por qué no basta con enviar un archivo

Una foto se puede enviar cien veces y tú sigues teniéndola. Con el dinero eso no puede ser: se pagaría a dos personas con la misma moneda. Esto se llama **doble gasto**.

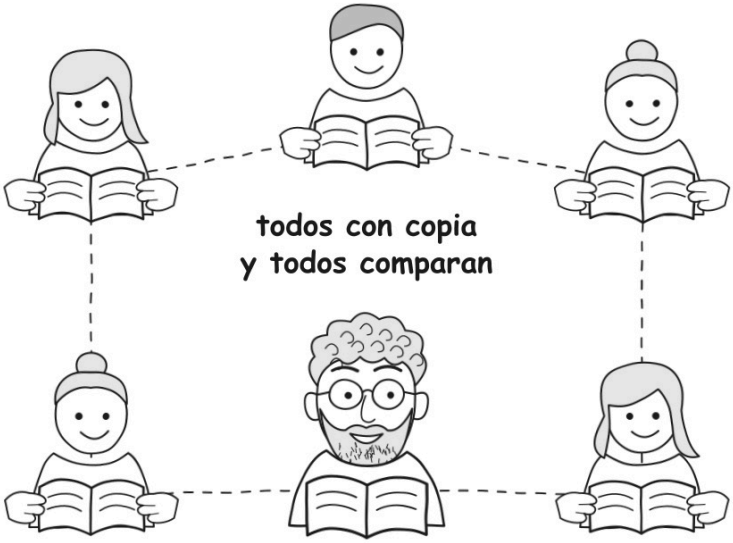


Durante mucho tiempo hubo una sola solución: alguien al mando lleva la lista y dice de quién es ahora cada moneda. Es decir, otra vez un intermediario.

La idea: los registros los guardan todos

En 2008, un autor bajo el seudónimo de Satoshi Nakamoto describió otro camino, y en enero de 2009 empezó a funcionar la red Bitcoin. La idea es esta:

1. La lista de transferencias no la esconde uno solo. **Muchos tienen una copia.**
2. Los participantes **comprueban con reglas comunes** cada transferencia nueva: si hay dinero y si no se ha gastado ya.
3. Los participantes **se ponen de acuerdo**, sin ningún jefe, sobre qué registros son válidos.



A esa lista común que se lleva entre todos se la llama **blockchain**. Cómo se enlazan los registros en una cadena lo veremos en el capítulo 5, y cómo se ponen de acuerdo los participantes, en el capítulo 8.

Probemos: el juego del cuaderno común

No hace falta código. Reúne a dos amigos y que cada uno tenga su hoja. Al principio todos anotan: Ania tiene 10 monedas, Boris 5 y Vera 0. Hay una sola regla: **no se puede dar más de lo que se tiene**.

N.º	Quién quiere transferir	¿Se acepta?	Ania	Boris	Vera
inicio			10	5	0
1	Ania → Vera 4				
2	Vera → Boris 1				
3	Boris → Ania 7				
4	Ania → Boris 6				

Rellenen las casillas vacías y después comparen las hojas. Si todos tienen lo mismo, acaban de ponerse de acuerdo sin ningún jefe. La respuesta está al final del capítulo.

Qué da la blockchain y qué no da

Da	No da
Transferencias directas, cualquier día y a cualquier hora	Un botón de «cancelar»
Una misma cuenta accesible desde distintos países	Un precio fijo de la moneda: la cotización puede cambiar mucho
Registros que cualquiera puede comprobar	Discreción: las transferencias de cualquier dirección están a la vista de todos
No hay que confiar a nadie la custodia de la lista	Un servicio de soporte que recupere una clave perdida

No todo da saltos: hay monedas que intentan mantenerse cerca de un dólar y se llaman **stablecoins**. Pero detrás de las más extendidas hay una empresa emisora, que puede congelar el dinero de una dirección. Los detalles, en el capítulo 9.

Los intermediarios no han desaparecido: exchanges, billeteras, proveedores. Pero no son iguales. Un exchange puede tener tus monedas en custodia. El proveedor de la primera parte no guarda dinero: solo transmite tus preguntas al nodo.

La trampa: «si no me gusta, lo cancelo»

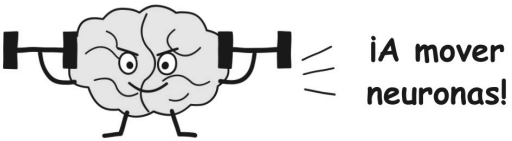
Una transferencia confirmada no se puede cancelar. Si te equivocaste de dirección, el dinero se fue. Solo puede devolverlo el destinatario, si quiere y si esa dirección es de alguien.



Por eso la dirección se comprueba dos veces, y antes de enviar una suma grande se prueba con una parte pequeña.

Una pequeña historia. En mayo de 2010 se pagaron dos pizzas con 10 000 bitcoins. Aquella transferencia sigue anotada en el cuaderno común.

Para pensar



1. Después del juego, Vera tiene 3 monedas. A la vez le escribe a Ania «te doy 3» y a Boris «te doy 3». ¿Qué deben hacer los participantes?
2. ¿Y si un participante aceptó la primera transferencia y otro la segunda? ¿Quién tiene razón?
3. ¿Qué le conviene más a Artyom: que un pago se pueda cancelar o que no? ¿Y a su cliente?

Pistas: 1) solo se puede aceptar una. 2) Es la pregunta más difícil; a ella se dedica el capítulo 8.

Lo esencial

- El dinero bancario son anotaciones, y alguien tiene que llevarlas.
- La blockchain es una lista común que muchos llevan con reglas compartidas.
- Librarse del intermediario también es una responsabilidad: no hay quien cancele una transferencia ni recupere una clave.

Respuesta del juego: la transferencia 1 se acepta (6, 5, 4); la 2 se acepta (6, 6, 3); la 3 se rechaza, Boris solo tiene 6 (6, 6, 3); la 4 se acepta (0, 12, 3). En total siguen siendo 15 monedas.

A continuación: capítulo 5. Qué es ese «bloque» que mencionó el nodo y por qué un registro antiguo no se puede reescribir a escondidas.

Capítulo 5. Bloques y cadena: por qué lo antiguo no se puede reescribir a escondidas

Aquel número

En el capítulo 3 el nodo le contestó a Artyom que el último bloque era el número 26 502 035. ¿Qué bloque es ese? ¿Por qué tiene número? ¿Y qué impide que alguien corrija en secreto una anotación del cuaderno?

Entendamos: un bloque es una página del cuaderno

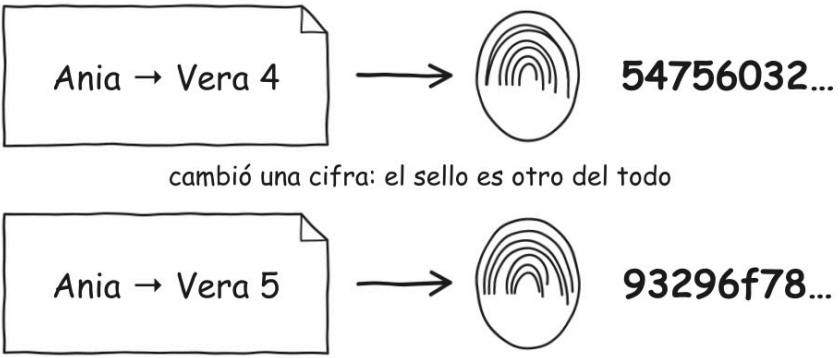
Las transferencias no se anotan de una en una. Se reúnen en tandas, como líneas en una página. Esa página se llama **bloque**. Contiene:

1. **Transferencias** y otras operaciones, llamadas transacciones.
2. El **número** de página, en orden.
3. La **hora** de creación.
4. **El sello de la página anterior.** Aquí está todo el secreto.

El sello, o hash

De cualquier texto se puede calcular una cadena corta de cifras y letras. Es el **hash**; nosotros lo llamaremos sello. Se parece a una huella digital:

- el mismo texto siempre da el mismo sello;
- si cambias aunque sea una letra, el sello es completamente distinto;
- a partir del sello no se puede reconstruir el texto;
- en la práctica es imposible encontrar otro texto con el mismo sello.



Es el comienzo de los hashes SHA-256 reales de las dos líneas del dibujo. SHA-256 es el que usa Bitcoin; otras redes calculan los sellos de otras formas.

La cadena

En cada página está anotado el sello de la anterior. La página obtiene su propio sello de todo lo que contiene, **incluido ese sello anterior**.

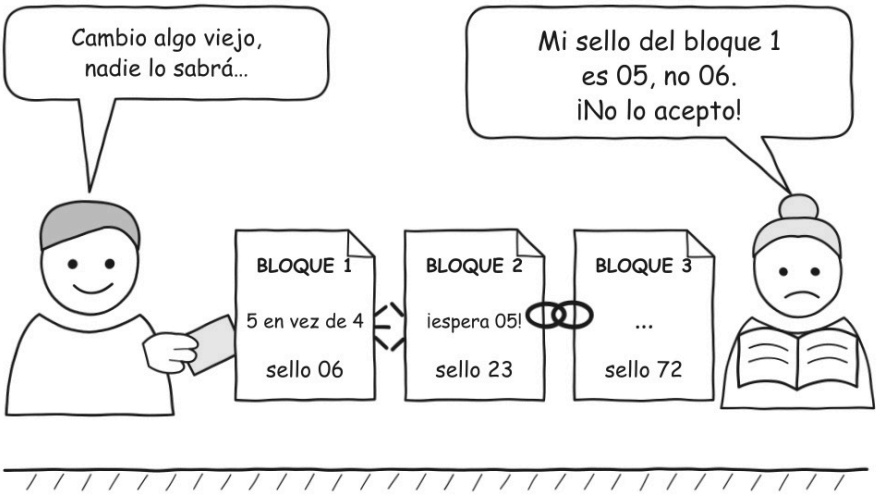


Así las páginas se enganchan en una **cadena de bloques**. En inglés, blockchain. Ese es todo el misterio del nombre.

Los sellos del dibujo son de juguete, de dos cifras: así se pueden calcular de cabeza.

Qué pasa si se retoca algo antiguo

Un estafador cambia un 4 por un 5 en el primer bloque. El sello del bloque cambia. Pero en el segundo bloque está anotado el antiguo. La cadena se rompe y se ve al instante.



Puede reescribir también todos los sellos siguientes. Pero los demás participantes tienen sus copias, y su versión no coincidirá con ellas. Y para que la aceptaran, tendría que imponerse al resto de la red. Es carísimo. Cómo decide exactamente la red qué cadena es la buena lo veremos en el capítulo 8.

Mira una cadena de verdad

Son tres bloques de la red Ethereum del 26 de noviembre de 2026. Los sellos son largos, así que se muestran el principio y el final.

Número de bloque	Su sello	Sello del anterior	Transacciones
26 502 033	`0xbe58…d764`	`0x2844…c876`	229
26 502 034	`0x7bb7…b7a4`	`0xbe58…d764`	229
26 502 035	`0xeab2…c904`	`0x7bb7…b7a4`	515

Sigue con el dedo: el sello de una fila se repite en la siguiente. Entre estos bloques pasaron exactamente 12 segundos. En Bitcoin los bloques nuevos aparecen con menos frecuencia, de media cada 10 minutos aproximadamente.

El primerísimo bloque no tiene anterior. El primer bloque de Bitcoin se creó el 3 de enero de 2009.

Probemos: tu propio sello

Inventemos una regla de juguete:

Sello = (sello anterior × 3 + todas las cantidades de la página); del resultado tomamos las dos últimas cifras.

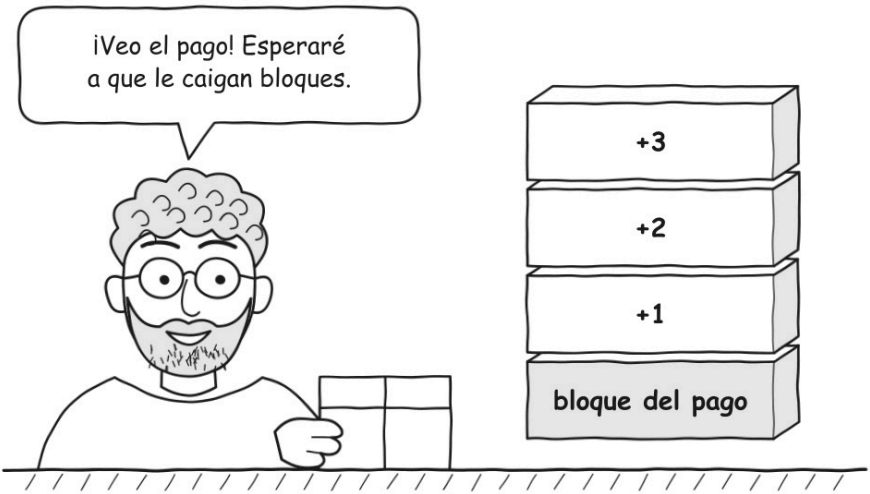
Bloque	Cantidades transferidas	Sello	Si en el bloque 1 se cambia 4 por 5
1	4 y 1		
2	6 y 2		
3	3		

Para el primer bloque, el sello anterior es 0. Rellena las dos columnas. ¿Cuántos sellos cambiaron?

La trampa: «si lo vi en un bloque, es para siempre»

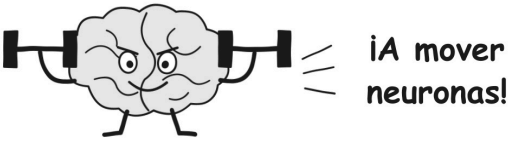
Los bloques más recientes a veces se sustituyen por otros: distintas partes de la red aún no se han puesto de acuerdo. Cuantos más bloques haya encima, más difícil es quitar tu anotación.

Esto se cuenta en **confirmaciones**: el propio bloque con el pago es la primera, y cada bloque siguiente suma una más.



Cuánto esperar depende de la red y de la cantidad. Por una taza de café, menos; por un pedido grande, más. Los detalles, en el capítulo 8.

Para pensar



1. Engaña a nuestro sello: cambia en el bloque 1 las cantidades 4 y 1 por otras de modo que el sello no cambie. ¿Qué dice eso de nuestra regla?
2. ¿Por qué es más difícil cambiar un bloque de hace tres años que el de ayer?
3. Si aparece un bloque cada 12 segundos, ¿cuántos bloques salen en una hora?

Respuestas: 1) por ejemplo, 3 y 2: la suma es la misma. Un hash de verdad lo habría notado. 2) Habría que rehacer todo lo que se apiló encima. 3) 300.

Lo esencial

- Un bloque es una página con transacciones, número, hora y el sello de la página anterior.
- Cambiar un registro antiguo cambia el sello y rompe la cadena.
- Se confía menos en un bloque reciente: se esperan confirmaciones.

Respuesta del juego: los sellos son 05, 23, 72. Tras el cambio: 06, 26, 81; cambiaron los tres.

A continuación: capítulo 6. ¿Cómo sabe la red que la transferencia la envió precisamente el dueño del dinero? Claves, direcciones y firma.

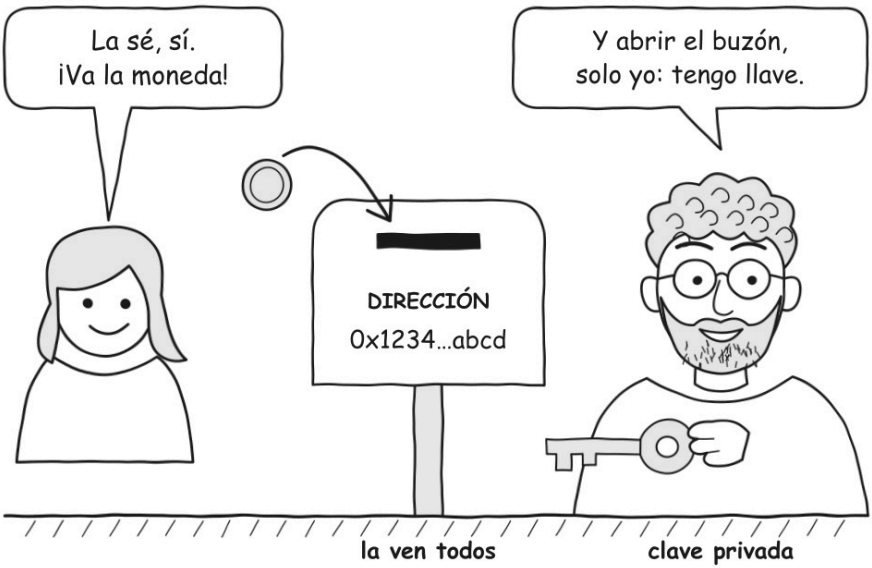
Capítulo 6. Claves, direcciones y firma

¿Cómo sabe la red que ese dinero es tuyo?

En el cuaderno común no hay pasaportes. Nadie pregunta el nombre ni coteja una foto. Entonces, ¿qué impide que cualquiera escriba: «Artyom me lo da todo»?

Entendamos: un buzón y una llave

Imagina un buzón en la calle. El número del buzón lo conoce todo el mundo, y cualquiera puede echar una moneda por la ranura. Pero abrirlo y sacarla solo puede quien tiene la llave.



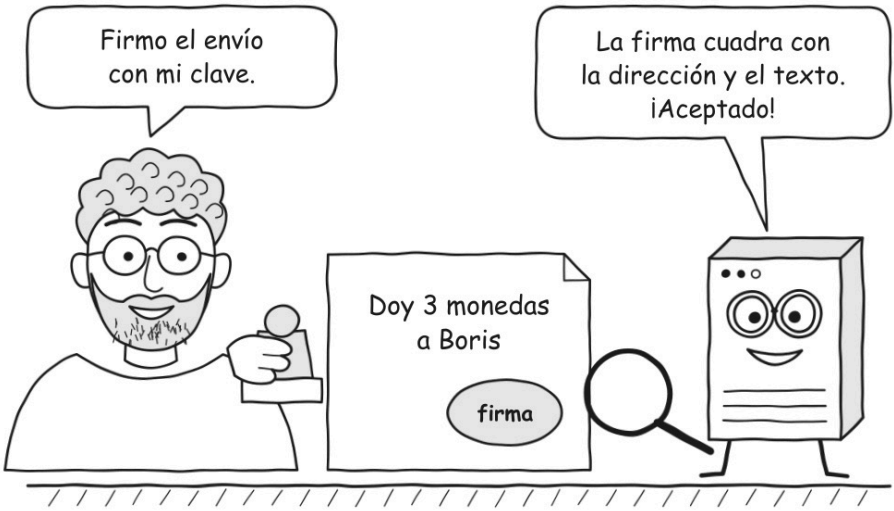
Qué	A qué se parece	A quién mostrarlo
Dirección	El número del buzón	A quien sea, para recibir dinero
Clave pública	La muestra con la que se comprueba la firma	Se puede a todos
Clave privada	La llave del buzón	**A nadie**

Están relacionadas en un solo sentido. De la clave privada se obtiene la pública, y de la pública, la dirección. En la práctica es imposible calcularlo al revés. En algunas redes la dirección es la propia clave pública, solo que escrita de una forma especial.

Una **billetera** es un programa o un dispositivo donde se guardan las claves. Dentro no hay monedas: siguen siendo anotaciones en el cuaderno común.

La firma

Para entregar monedas, el dueño redacta la transferencia y la **firma** con su clave privada. La clave no se envía a ninguna parte. A la red van la transferencia y la firma.



El nodo comprueba dos cosas: que la firma se hizo con la clave de esa dirección precisamente y que el texto no cambió después de firmarlo. La firma de una transferencia no sirve para otra. Y la red no ejecutará esa misma transferencia por segunda vez.

Probemos: una firma de juguete

Todo se calcula con una calculadora. La clave privada es el número **7** y la pública, el **3**. Solo la primera es secreta.

> Regla: multiplica por el número elegido y, después de cada paso, quédate con el resto de dividir entre 33.

Firmamos el mensaje «4». Tomamos el 4 tantas veces como mande la clave privada, es decir, 7:

Cuántos cuatros	1	2	3	4	5	6	7
Resto	4	16	31	25	1	4	**16**

Cada paso es el resto anterior multiplicado por 4. Por ejemplo, el tercero: 16 × 4 = 64, y el resto de dividir entre 33 es 31. Después del séptimo paso obtenemos la firma: **16**.

Comprobamos. Tomamos la firma 16 tantas veces como mande la clave pública, es decir, 3: 16; luego 16 × 16 = 256, resto 25; luego 25 × 16 = 400, resto **4**. Ha salido nuestro mensaje. ¡La firma es auténtica!

Ahora alguien cambia el mensaje a «5» y deja la firma 16. La comprobación sigue dando 4, no 5. La falsificación se ve.

Esto es solo la idea. Unos números tan pequeños los rompe un escolar. En las redes de verdad las claves son números enormes y las matemáticas son otras.

La frase secreta

La clave privada es incómoda de copiar: es una cadena larga. Por eso muchas billeteras dan una **frase secreta** de 12 o 24 palabras. Con ella se recuperan las claves y todas las direcciones.



- Quien conoce la frase es el dueño del dinero.
- Si pierdes la frase y el dispositivo, no hay quien lo recupere.
- No la fotografíes ni la guardes en notas o chats.
- No la escribas en sitios web ni la dictes al «soporte». El soporte de verdad no la pide.

Una contraseña se puede cambiar; una clave privada, no. Si se filtró, solo queda transferir el dinero cuanto antes a una dirección nueva.

Qué significa esto para una tienda

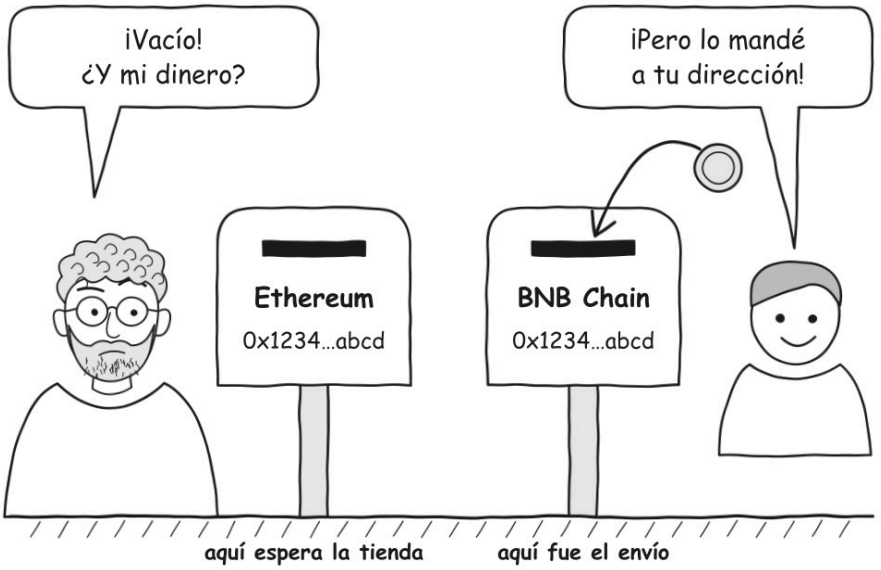
Para **recibir** dinero y ver el pago, el sitio de Artyom solo necesita la dirección. La clave privada no hace falta en el servidor de la tienda, y no conviene tenerla ahí: si hackean el sitio, se llevan también el dinero. Es cómodo dar a cada pedido su propia dirección; de eso trata el capítulo 28.

La trampa: «la dirección es la misma, así que llegará»

Las direcciones de distintas redes tienen distinto aspecto:

Red	La dirección empieza por
Bitcoin	`1`, `3` o `bc1`
Ethereum y redes parecidas	`0x`
TRON	`T`
TON	normalmente `EQ` o `UQ`
Solana	no tiene un comienzo especial

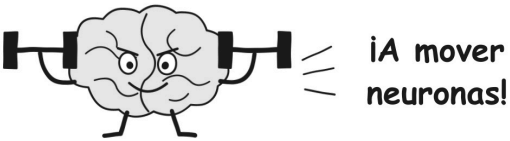
Pero en Ethereum, BNB Smart Chain, Polygon y muchas otras redes la dirección tiene un aspecto **idéntico**. Y sus cuadernos son distintos. Si el cliente envió por la red equivocada, la tienda no verá el pago.



Por eso en la factura se indica no solo la dirección y la cantidad, sino también la **red**.

Y otro truco de los estafadores: envían una transferencia minúscula desde una dirección parecida a una que conoces. Cuentan con que la copies del historial. Coteja la dirección entera, no solo los primeros y los últimos caracteres.

Para pensar



1. Firma el mensaje «2» con la clave 7 y comprueba la firma con la clave 3.
2. Artyom imprimió la dirección de la tienda en su tarjeta de visita. ¿Es peligroso?
3. El «soporte de la billetera» pide la frase secreta para recuperar una transferencia atascada. ¿Qué hacer?

Respuestas: 1) la firma es 29 y la comprobación vuelve a dar 2. 2) Así no robarán el dinero, pero todos verán los ingresos de esa dirección. 3) No contar nada: son estafadores.

Lo esencial

- La dirección se muestra a todos; la clave privada y la frase secreta, a nadie.
- La firma demuestra que la transferencia la autorizó el dueño de la clave y que no se ha modificado.
- La red es tan importante como la dirección.

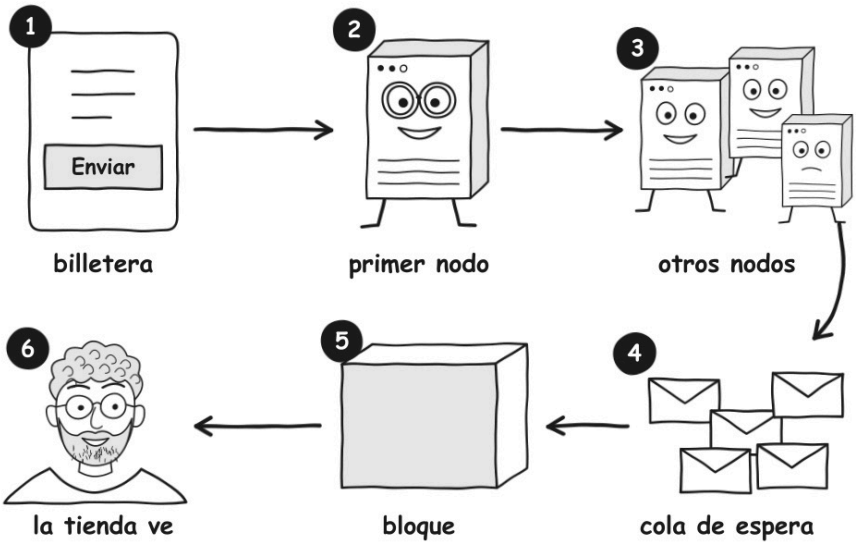
A continuación: capítulo 7. La transferencia está firmada. ¿Qué le pasa desde el botón «Enviar» hasta quedar anotada en un bloque?

Capítulo 7. El viaje de una transacción: del botón «Enviar» al bloque

«¡Ya lo envié!» — «Pues aquí no hay nada»

El cliente pulsó el botón y escribe: «Enviado». Artyom mira y el dinero no está. ¿Alguien miente? Lo más probable es que nadie. La transferencia sigue de camino.

Entendamos: seis paradas



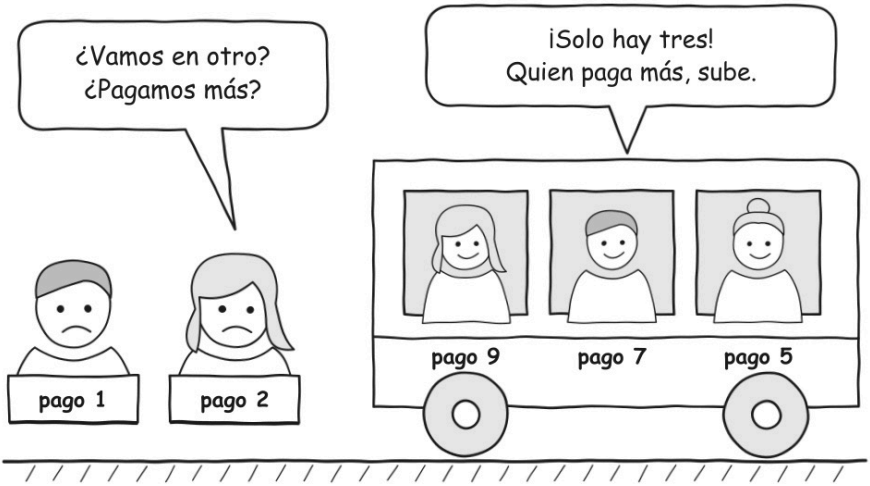
1. La **billetera** redacta la transferencia y la firma con la clave.
2. El **primer nodo** la recibe y comprueba si la firma es correcta y si alcanza el dinero.
3. La reenvía a **otros nodos**, y estos, a otros más.
4. La transferencia espera en la **cola**. En inglés se llama mempool.
5. Aquel a quien le toca armar la página mete la transferencia en el **bloque**.
6. La **tienda** la ve y espera las confirmaciones.

Los nombres y los detalles cambian de una red a otra: en algunas, por ejemplo, no hay cola común y la transferencia va directamente a quien arma el bloque.

Cada transferencia tiene su propio «sello»: el **hash de la transacción**. Es como el número de seguimiento de un paquete: con él se puede saber dónde está ahora la transferencia.

La comisión: asientos en el autobús

En un bloque no cabe todo. Imagina un autobús con tres asientos y cinco personas en la parada.



Para entrar en un bloque, quien envía paga una **comisión** de la red. Cuantos más interesados hay, más caro es el asiento. Quien ofreció poco espera el autobús siguiente, y a veces no llega a subir nunca.

- La comisión la paga quien envía, y no se la queda la tienda.
- No depende de la cantidad transferida, sino de su «peso» y de lo ocupada que esté la red.
- Cómo se calcula ese «peso» cambia en cada red: de eso trata la cuarta parte del libro.

Mira una transferencia de verdad

Es una transferencia normal en la red Ethereum, bloque 26 502 085. Se le preguntó por ella al nodo usando el hash de la transacción:

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

La respuesta es larga. Esto es lo principal, convertido a números corrientes:

Campo	Valor	Qué significa
`status`	`0x1`	Ejecutada con éxito
`blockNumber`	26 502 085	En qué bloque está
`gasUsed`	21 000	El «peso» de una transferencia simple
comisión	0,000043 ETH	«Peso» × precio por unidad
cantidad transferida	0,000041 ETH	Se toma de la propia transferencia

¿Te has fijado? Esta vez se pagó por enviar **más de lo que se envió**. A la comisión le da igual cuánto dinero va dentro.

Y esta cifra cambia todo el tiempo. Pregúntalo tú:

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

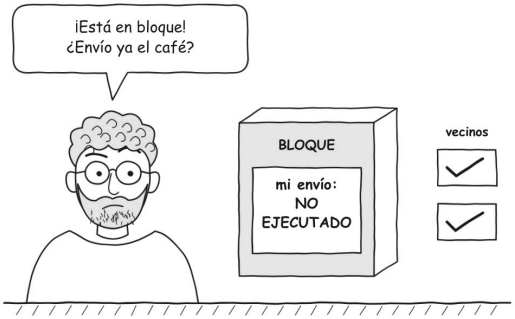
El nodo dirá el precio actual de una unidad de «peso» en wei. Y en el nodo público de Bitcoin de RpcNode, ese mismo día esperaban en la cola casi 83 000 transferencias.

Qué le puede pasar a una transferencia

Estado	Qué significa	Qué debe hacer la tienda
Esperando	Sigue en la cola	Esperar, no enviar el producto
En un bloque	Registrada, pero con pocas confirmaciones	Comprobar que tuvo éxito y esperar
Confirmada	Tiene suficientes bloques encima	Enviar el pedido
Fallida	Está en un bloque, pero la transferencia no se realizó	No contarla como pago
Salió de la cola o fue reemplazada	No entró en ningún bloque	Pedir al cliente que revise su billetera

La trampa: «si está en un bloque, está pagado»

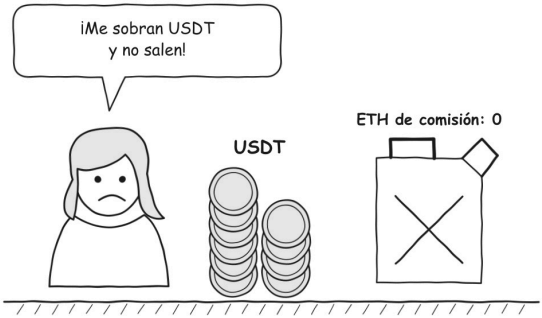
En Ethereum y redes parecidas, una transferencia puede entrar en un bloque y aun así no ejecutarse. Por ejemplo, al cliente no le alcanzaban los tokens. La comisión se cobra igual, pero el dinero no te llega.



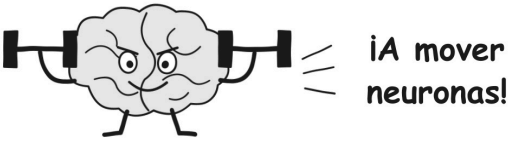
Por eso no se mira solo el bloque, sino también el resultado: en el ejemplo de arriba es el campo `status`. Y además se cotejan el destinatario, la cantidad, la moneda y la red.

Segunda trampa: el depósito vacío

La comisión se cobra en la moneda propia de la red. En Ethereum es ETH, aunque estés enviando USDT. En TRON se paga con TRX y con recursos especiales de la red.



Para pensar



1. En un bloque caben tres transferencias. En la cola ofrecen por un asiento 4, 9, 1, 7 y 2. ¿Quiénes viajan en el primer bloque?
2. El cliente mandó el hash de la transacción y una captura de pantalla que dice «éxito». ¿Basta para enviar el café?
3. El cliente tiene 100 USDT en la red Ethereum y ni una gota de ETH. ¿Qué debe hacer?

Respuestas: 1) 9, 7 y 4. 2) No: una captura se puede falsificar; comprueba el hash por tu lado. 3) Primero conseguir un poco de ETH para la comisión.

Lo esencial

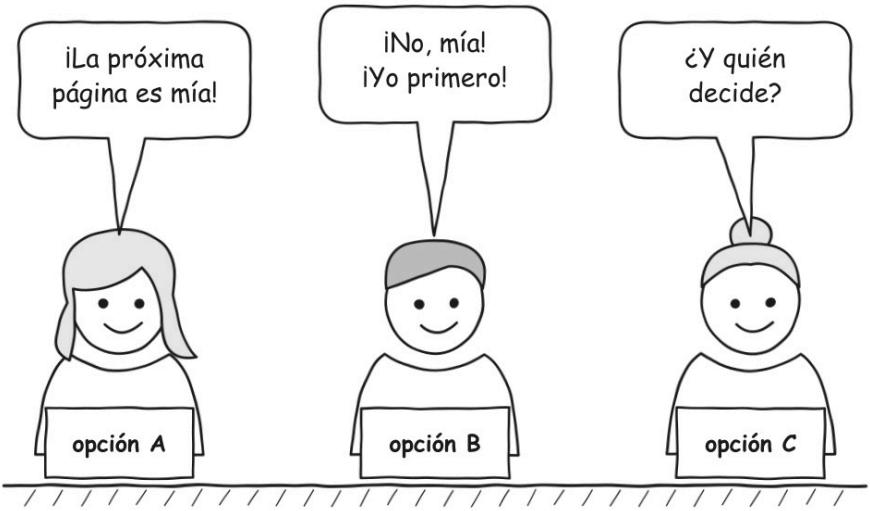
- Entre «enviado» y «recibido» hay un camino: nodo, cola, bloque y confirmaciones.
- La comisión compra un sitio en el bloque y no depende de la cantidad.
- Un pago es una transferencia con éxito de la moneda correcta, en la red correcta y con confirmaciones suficientes.

A continuación: capítulo 8. ¿Quién arma los bloques y cómo se ponen todos de acuerdo en que esta cadena precisamente es la buena?

Capítulo 8. Cómo se pone de acuerdo la red

¿Quién escribe la página siguiente?

En el cuaderno común no manda nadie. Miles de participantes tienen una copia y cualquiera puede proponer su propia página siguiente. ¿Cuál se considera la auténtica?



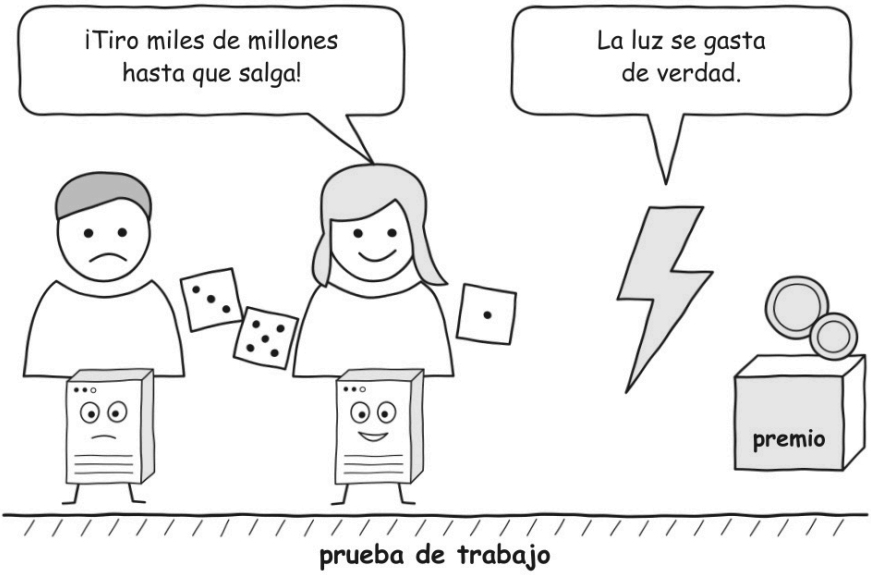
Entendamos: el consenso

Las reglas con las que los participantes llegan a una respuesta común se llaman **consenso**. Su misión es que engañar salga más caro que jugar limpio.

No basta con contar votos: un estafador pondría en marcha un millón de programas y tendría un millón de votos. Por eso el voto debe costar algo. Hay dos formas principales.

Forma 1: prueba de trabajo

En inglés, Proof of Work, y al proceso se le llama **minería**. Los participantes compiten por resolver un problema que no se puede resolver con ingenio, solo probando. Es como tirar dados hasta que sale una combinación rara.

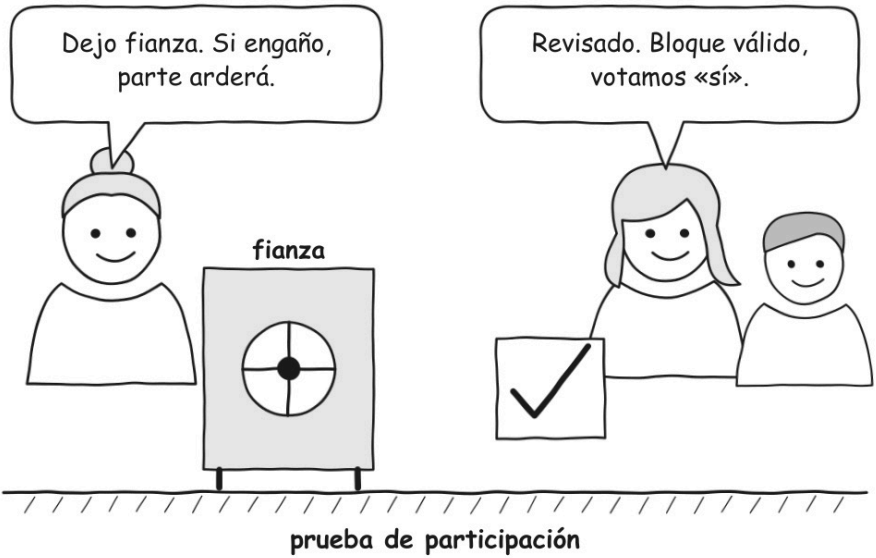


El ganador añade el bloque y recibe una recompensa: monedas nuevas y comisiones. Comprobar su respuesta es fácil; encontrarla, caro. Así funciona Bitcoin:

- aparece un bloque de media cada 10 minutos;
- cada 2016 bloques se ajusta la dificultad del problema para mantener ese ritmo;
- ahora se dan 3,125 BTC en monedas nuevas por bloque, y aproximadamente cada cuatro años la recompensa se reduce a la mitad.

Forma 2: prueba de participación

En inglés, Proof of Stake, o **staking**. En lugar de electricidad, el participante deja monedas en garantía. Se le llama **validador**. Los validadores proponen bloques por turnos y los demás los comprueban y votan.

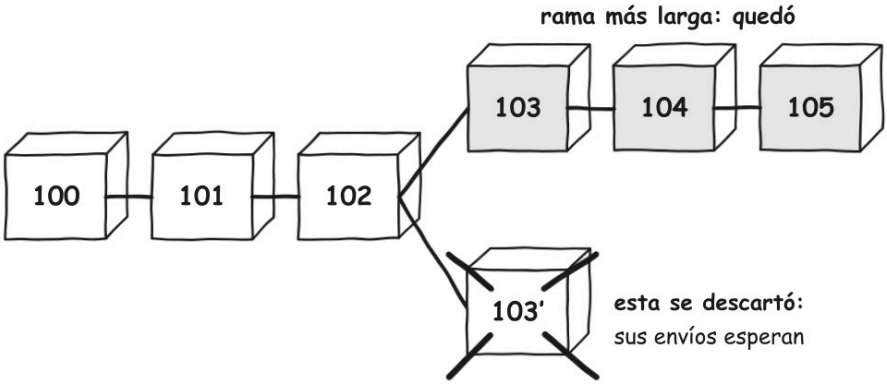


Por una infracción grave se quema parte de la garantía; se llama slashing. Ethereum pasó a este método el 15 de septiembre de 2022, y el consumo eléctrico de la red se redujo en torno a un 99,95%. Para ser validador por tu cuenta hacen falta 32 ETH.

También hay variantes. En TRON escriben los bloques 27 representantes elegidos por los poseedores de monedas. En BNB Smart Chain también trabaja un círculo limitado de validadores.

Cuando salen dos páginas a la vez

La red es grande y las noticias no llegan al instante. A veces aparecen dos bloques con el mismo número y la cadena se bifurca durante un tiempo. Luego una rama gana y la otra se descarta. Esto se llama **reorganización**.



Una transferencia de un bloque descartado no desaparece sin dejar rastro: normalmente vuelve a la cola y entra en otro bloque. Pero puede que no entre. Para eso sirven las confirmaciones del capítulo 5.

En el dibujo gana la rama larga. Para ser exactos, en Bitcoin se elige la que tiene más trabajo invertido, y en las redes con staking, la que reúne más votos respaldados por garantía.

Finalidad: cuando ya no se anula

La **finalidad** es el momento a partir del cual un bloque ya no se sustituye. En algunas redes se anuncia de forma explícita; en otras, la certeza simplemente crece con cada bloque.

Red	Cuándo se considera definitivo, aproximadamente
Bitcoin	No hay marca explícita. Para sumas grandes se aconsejan 6 confirmaciones, cerca de una hora
Ethereum	Normalmente 13–15 minutos, a veces más
TRON	Normalmente cerca de un minuto
BNB Smart Chain	Cerca de un segundo o menos
Solana	Tras la actualización de 2026 está pensada para fracciones de segundo
TON	Cerca de un segundo

Datos a 27 de noviembre de 2026 según la documentación de las redes. Estas cifras cambian con las actualizaciones.

Probemos: pregunta tres niveles de «frescura»

A los nodos de Ethereum se les puede pedir no solo el bloque más nuevo. La palabra `latest` del capítulo 1 tiene dos parientes: `safe` (ya fiable) y `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]}'
```

Cambia `finalized` por `safe` y `latest` y compara los números. Cuando se escribía este capítulo, el nodo contestó así:

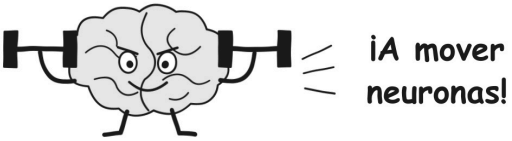
Qué se preguntó	Número de bloque	Atraso respecto al más reciente
`latest`	26 502 104	—
`safe`	26 502 070	34 bloques, unos 7 minutos
`finalized`	26 502 038	66 bloques, unos 13 minutos

La trampa: «en todas las redes es igual»

«Espero una confirmación y ya está» es una regla peligrosa. Una confirmación significa en cada red una fiabilidad muy distinta. Decide según la red y la cantidad:

- cuanto más caro sea el pedido, más hay que esperar;
- para sumas grandes, espera la finalidad donde la haya;
- no entregues algo que no se pueda recuperar basándote en una transferencia de un bloque recién creado.

Para pensar



1. ¿Por qué en el cuaderno no se puede votar según el principio «un programa, un voto»?
2. Los bloques de Bitcoin salen cada 10 minutos. ¿Cuánto hay que esperar de media 6 confirmaciones? ¿Y 3?
3. Un cliente compra una taza de café para llevar y otro compra 200 kg de grano. ¿Con cuál debe Artyom esperar la finalidad?

Respuestas: 1) los programas no cuestan nada y se pueden lanzar cuantos se quiera. 2) Cerca de una hora y cerca de media hora. 3) Con el que compra 200 kg.

Lo esencial

- El consenso son las reglas con las que los participantes, sin ningún jefe, eligen una sola cadena.
- El voto cuesta trabajo (minería) o garantía (staking).
- Los bloques recientes pueden sustituirse; lo definitivo llega más tarde y de forma distinta en cada red.

A continuación: capítulo 9. Monedas, tokens y contratos inteligentes: por qué USDT no es lo mismo que ETH.

Capítulo 9. Monedas, tokens y contratos inteligentes

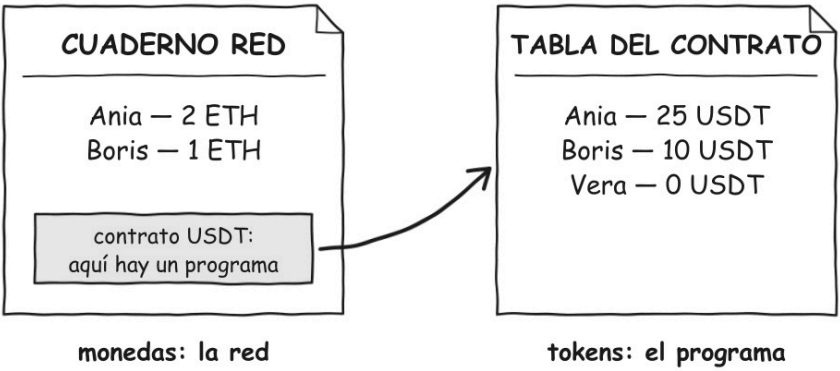
«Yo pagué y a ustedes no les ha llegado nada»

El cliente envió 25 USDT en la red Ethereum. Artyom pregunta al nodo, como en el capítulo 1, y el saldo no ha cambiado. ¿Se perdió el dinero? No. Es que había que preguntar de otra manera.

Entendamos: moneda y token

La **moneda nativa** es la que lleva la propia red: ETH en Ethereum, TRX en TRON, BNB en BNB Smart Chain. Con ella se paga la comisión.

Un **token** son anotaciones que lleva un programa que vive dentro de la red. USDT es un token.

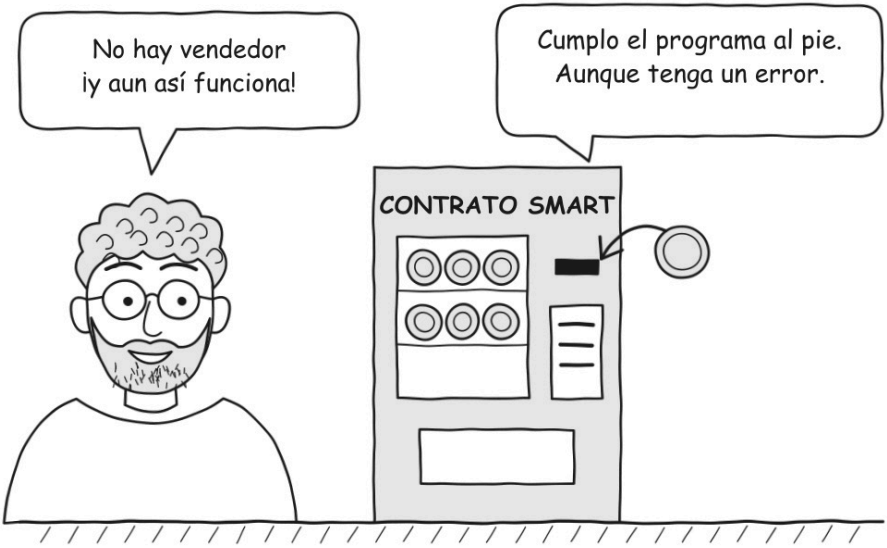


	Moneda nativa	Token
Quién lleva las anotaciones	la propia red	un contrato inteligente
Ejemplo en Ethereum	ETH	USDT, USDC
Con qué se paga la comisión	con ella misma	con la moneda nativa de la red
Cómo preguntar el saldo	`eth_getBalance`	`eth_call` al contrato

En cada red los tokens funcionan de un modo distinto. Aquí vemos Ethereum y las redes parecidas.

Contrato inteligente: una máquina sin vendedor

Un **contrato inteligente** es un programa escrito en la blockchain. Se parece a una máquina expendedora: metes algo, pulsas y recibes un resultado según reglas fijadas de antemano. Todos los nodos lo ejecutan de la misma forma.



- No es un acuerdo con firmas, sino literalmente un programa.
- Normalmente no se puede cambiar, pero algunos autores se reservan esa posibilidad.
- El programa puede tener un «dueño» con derechos especiales.
- Un error en el programa puede costar dinero, y no hay quien lo deshaga.

Probemos: el saldo de un token

El método `eth_call` significa «pregúntale al contrato sin cambiar nada». La red no cobra comisión por esa pregunta: no se escribe nada en el cuaderno. Preguntemos al contrato de USDT cuántos USDT hay en su propia dirección:

```
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` es el contrato al que preguntamos. Y `data` está pegada a partir de tres trozos:

Trozo	Qué significa
`0x70a08231`	qué pregunta: «¿cuál es el saldo?» (`balanceOf`)
24 ceros	relleno hasta la longitud necesaria
40 caracteres	de quién es el saldo: la dirección sin `0x`

Cuando se escribía este capítulo, el nodo contestó así:

```
{"jsonrpc":"2.0","id":1,"result":"0x000000000000000000000000000000000000000000000000
014d0a9aa6c5"}
```

Después de los ceros está `14d0a9aa6c5`, que es 1 430 402 016 965. El USDT de Ethereum tiene seis decimales, así que en la dirección del contrato hay **1 430 402,016965 USDT**.

¿De dónde han salido? Todo indica que son sobre todo transferencias «a donde no era»: alguien pegó la dirección del contrato en lugar de la del destinatario. No hay una forma normal de que quien los envió los recupere.

Dónde va la coma

El contrato solo guarda números enteros. Cuántas cifras separar lo dice una propiedad aparte: `decimals`.

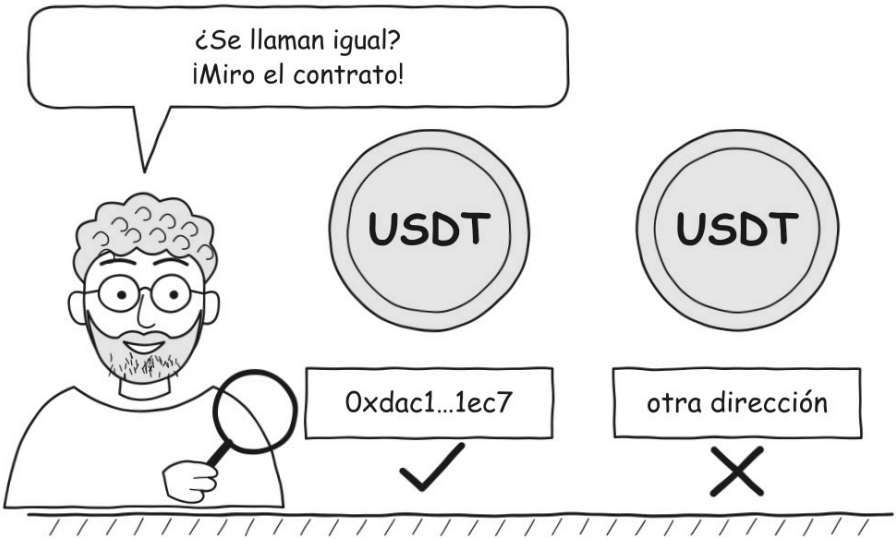
Qué	Decimales
ETH, la moneda nativa	18
USDT en Ethereum	6
token «Tether USD» (USDT) en BNB Smart Chain	18

El nombre es el mismo, pero hay el triple de decimales. Si los confundes, te equivocas por un billón de veces. Además, la dirección de ese token de BNB Smart Chain no figura entre los contratos de USDT en el sitio de la empresa Tether: un mismo nombre no significa una misma emisión.

Comprobado el 27 de noviembre de 2026 con consultas a los nodos públicos y en la página tether.to/en/supported-protocols.

La trampa: el token falso

Cualquiera puede llamar «USDT» a su token. El nombre no demuestra nada. El USDT auténtico se reconoce por una pareja: **la red y la dirección del contrato**.



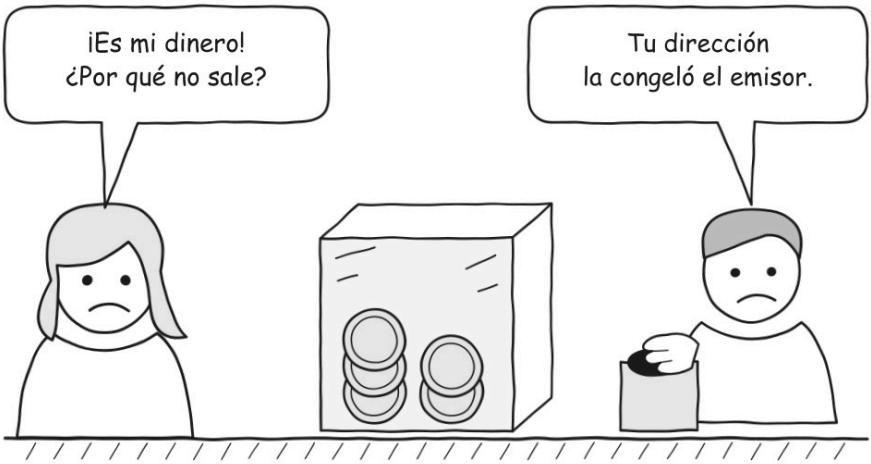
Por eso la tienda lo comprueba todo a la vez:

- la red;
- la dirección del contrato del token;
- el destinatario;
- la cantidad con el número correcto de decimales;
- el éxito y las confirmaciones.

Stablecoins: cómodas, pero tienen dueño

Una **stablecoin** intenta mantenerse cerca de un dólar. En el caso de USDT es una promesa de la empresa emisora, no una ley de la naturaleza.

El emisor además tiene poder: las condiciones de uso de Tether recogen expresamente el derecho a congelar tokens (versión del 26 de febrero de 2026, tether.to/en/legal).

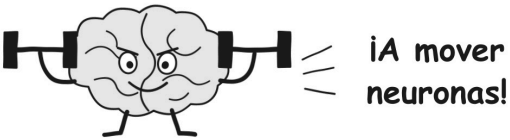


Segunda trampa: «autoriza el gasto»

Un sitio puede pedirte que firmes una **autorización** (`approve`): «este contrato puede gastar mis tokens». Si autorizas a un estafador sin límite de cantidad, se los llevará más tarde, sin ninguna firma nueva. No necesita conocer tu frase secreta.

> **Regla:** si no entiendes lo que firmas, no firmes.

Para pensar



1. Para el USDT de Ethereum, el nodo devolvió el número 25 000 000. ¿Cuántos USDT son?
2. El cliente envió un token llamado USDT, pero con otra dirección de contrato. ¿Está pagado el pedido?
3. Artyom tiene 100 USDT en Ethereum y 0 ETH. ¿Puede enviárselos a su proveedor de café?

Respuestas: 1) 25. 2) No. 3) No: no tiene con qué pagar la comisión, ver el capítulo 7.

Lo esencial

- La moneda la lleva la red; el token, un contrato inteligente.
- Un token se reconoce por la red y la dirección del contrato, no por el nombre.
- La cantidad se lee teniendo en cuenta los decimales.

Fin de la segunda parte. Artyom ya sabe qué son un bloque, una clave, una comisión y un token. Pero el nodo sigue siendo para él una caja negra. **A continuación:** parte 3, capítulo 10. Qué tipos de nodos hay.

Capítulo 10. Qué tipos de nodos hay

«¿Y cuánto había en aquel entonces?»

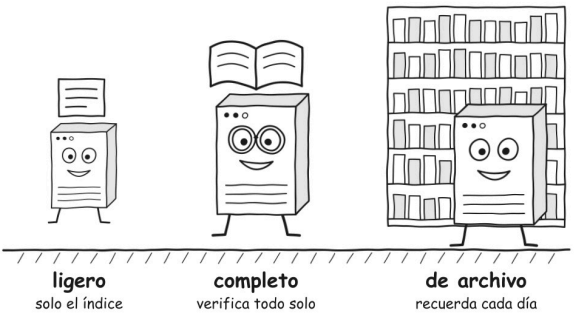
Artyom está cuadrando las cuentas y quiere saber cuánto había en la dirección de la tienda hace unos meses. El nodo responde de maravilla sobre hoy, pero sobre el pasado se niega. ¿Acaso la blockchain no lo guarda todo?

Entendamos: anotaciones y saldos

Aquí se esconden dos cosas distintas:

- **La historia:** las propias anotaciones, es decir, bloques y transferencias.
- **El estado:** quién posee qué en un momento concreto, es decir, saldos y datos de los contratos.

Es como en un banco: la lista de operaciones existe, pero el saldo de cada minuto del año pasado no está guardado aparte. Habría que recalcularlo. Los nodos se diferencian en cuánto de todo esto recuerdan.



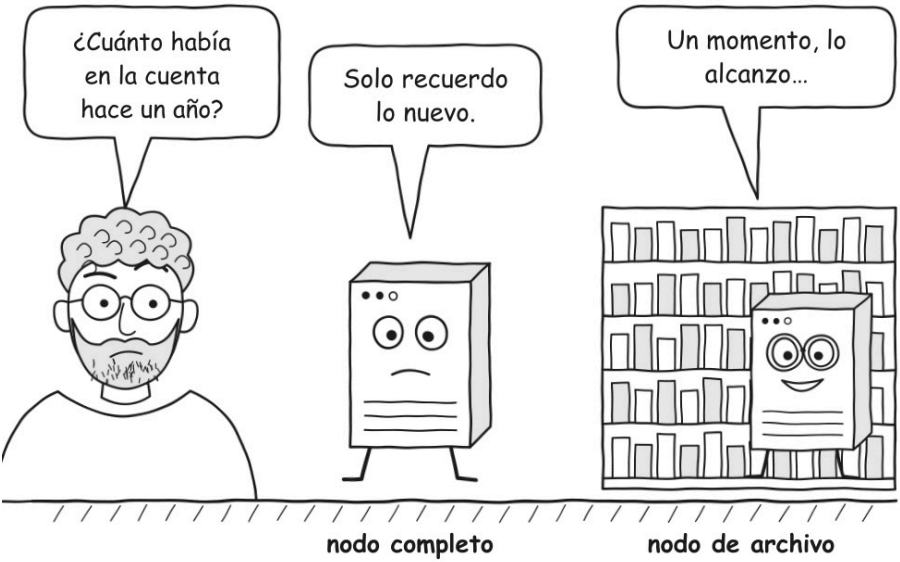
Tres tipos principales

Un **nodo completo** comprueba los bloques según las reglas de la red y guarda el estado reciente. En Ethereum suelen ser unos 128 últimos bloques, es decir, unos 25 minutos. La profundidad exacta depende del programa y de los ajustes.

Un **nodo de archivo** lo comprueba todo desde el principio y no borra lo que descargó. Recuerda el estado en cualquier bloque, pero a cambio necesita mucho más disco.

Un **nodo ligero** descarga solo las cabeceras de los bloques y los detalles se los pregunta a otros. La idea es buena, pero en Ethereum, a 27 de noviembre de 2026, no hay un nodo ligero listo

para el trabajo diario.



El nodo podado: comprobó y tiró

Bitcoin tiene un modo de ahorro de disco. Un **nodo podado** (pruned) lo comprueba todo y después borra los bloques antiguos. Sigue siendo un nodo completo de pleno derecho; simplemente no puede entregar lo que borró. Según los desarrolladores de Bitcoin Core, le bastan incluso 10 GB.

Ethereum también ha incorporado esta posibilidad: desde julio de 2025, los programas de los nodos pueden dejar de guardar el contenido de los bloques más antiguos, los anteriores al paso al staking en septiembre de 2022. Es un ajuste y no está activado en todas partes. Por eso hasta un nodo «completo» puede no recordar las anotaciones muy antiguas.

Probemos: preguntar por el pasado

Preguntemos dos veces el saldo de una misma dirección. Primero, el de ahora:

```
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` son 42 wei, unas migajas. Ahora cambia `"latest"` por `"0xe4e1c0"`, es decir, por el bloque número 15 000 000. El nodo público contestará así:

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

En español: «los datos de archivo no están disponibles en el endpoint público, usa uno personal». No es una avería, sino el límite del servicio gratuito. En RpcNode el acceso de archivo está en los planes de pago, y en Free no lo hay (capítulo 3).

Cuánto pesa esto

Nodo	Disco, aproximado
Bitcoin, podado	desde 10 GB
Bitcoin, completo	unos 900 GB
Ethereum, completo	unos 2 TB o más
Ethereum, de archivo	de 2 a 14 TB o más, según el programa

Los datos de Ethereum son de la página de ethereum.org sobre cómo ejecutar nodos, a 27 de noviembre de 2026.

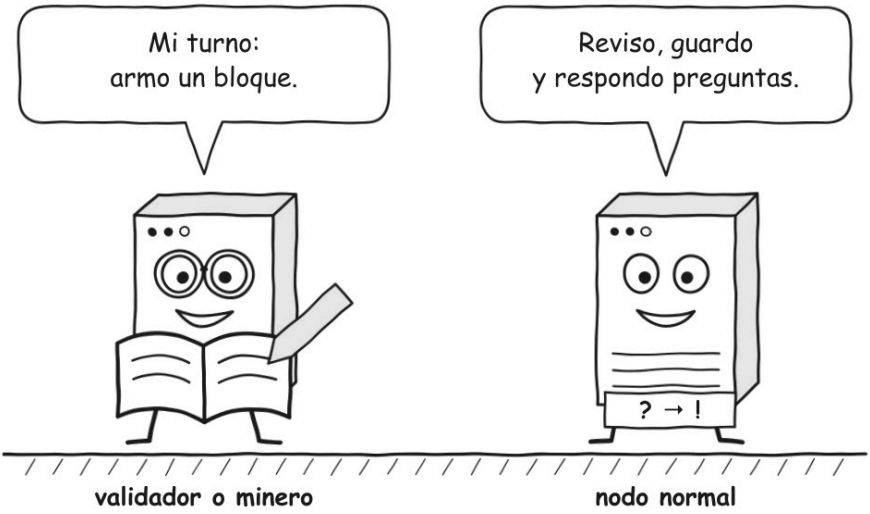
La segunda fila la puedes comprobar tú: el nodo de Bitcoin dice honradamente cuánto ocupa.

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

En la larga respuesta busca dos líneas. Cuando se escribía este capítulo decían: `"size_on_disk":896027659764` (unos 896 GB contando los archivos auxiliares) y `"pruned":false`: el nodo no está podado.

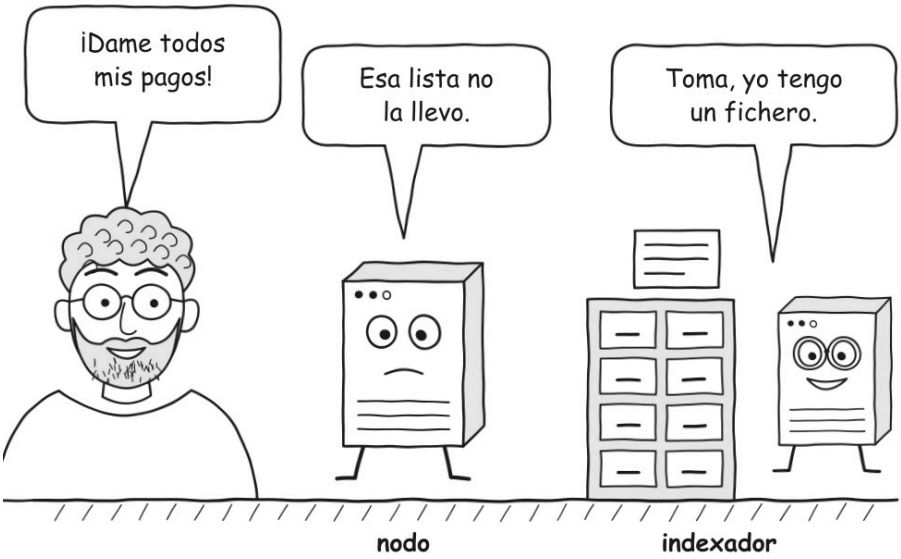
¿Y quién escribe los bloques?

Es un papel aparte. El **validador** o el **minero** participa en la creación de bloques (capítulo 8). Para comprobar pagos no hace falta serlo: basta con un nodo que lea, compruebe y responda.



La trampa: «le pediré al nodo todos mis pagos»

En el repertorio estándar de preguntas a un nodo de Ethereum no existe «muéstrame todas las transferencias de esta dirección». Ni siquiera un nodo de archivo sabe hacerlo. El método `eth_getTransactionCount` solo dirá el número de transacciones **enviadas** desde la dirección.

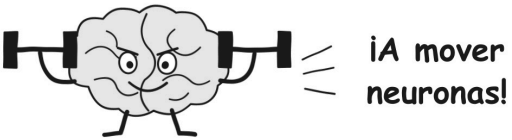


Para esas listas hace falta un **indexador**: un programa que lee los bloques y reparte las anotaciones en un fichero. Así funcionan también los sitios exploradores de blockchain. A una tienda le resulta más sencillo anotar sus pagos desde el principio y vigilar los nuevos; de eso trata el capítulo 26.

Qué necesitas para tu tarea

Tarea	Qué sirve
Comprobar un pago reciente	un nodo completo normal
Saber el saldo en una fecha pasada	un nodo de archivo
Obtener la lista de todas las transferencias de una dirección	un indexador
Participar en la creación de bloques	un validador o un minero

Para pensar



1. Artyom quiere trazar un gráfico del saldo de la tienda durante un año, con un punto por cada día. ¿Qué nodo necesita?
2. Un nodo podado de Bitcoin borró los bloques antiguos. ¿Se le puede confiar la comprobación de los nuevos?
3. ¿Qué hace falta para mostrar al cliente todos sus pedidos anteriores con sus pagos: un nodo de archivo o la base de datos de la tienda?

Respuestas: 1) de archivo. 2) Sí, lo comprobó todo antes de borrar. 3) La base de datos propia: el nodo no lleva esas listas.

Lo esencial

- La historia son las anotaciones; el estado, los saldos en un momento concreto.
- El nodo completo recuerda el estado reciente; el de archivo, cualquiera.
- El nodo no da la lista de transferencias de una dirección; para eso hace falta un indexador.

A continuación: capítulo 11. Ya le hemos hecho al nodo una decena de preguntas. Toca entender en qué idioma: JSON-RPC, REST, gRPC y WebSocket.

Capítulo 11. Cómo se habla con un nodo

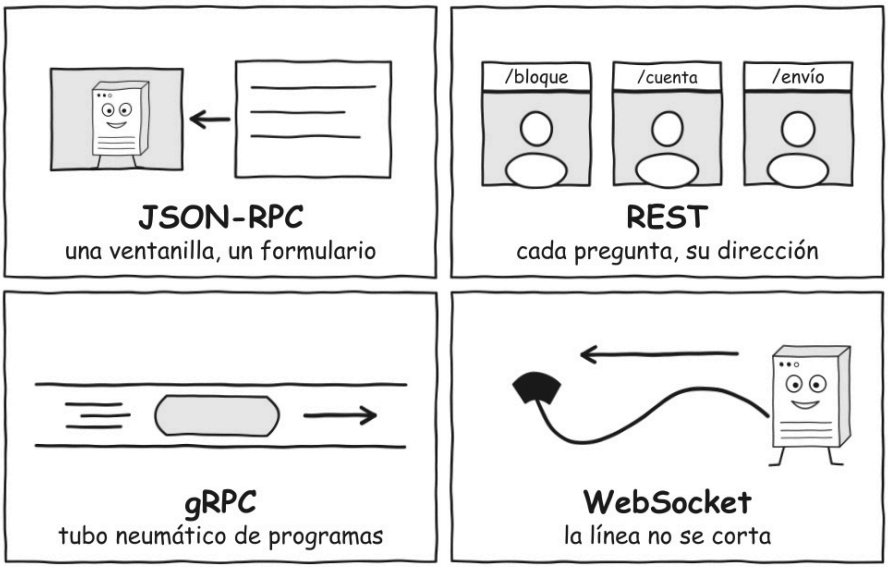
¿Qué son todas esas letras de las consultas?

Artyom ya ha copiado una decena de veces los comandos de este libro, y funcionaron. Es hora de entender qué dicen: qué es RPC, por qué hay tantas llaves y por qué en la documentación de las distintas redes las consultas tienen un aspecto diferente.

Entendamos: qué son API y RPC

Una **API** es una ventanilla por la que un programa se dirige a otro según reglas conocidas de antemano. **RPC** (llamada a procedimiento remoto) es una forma de pedir a un programa de otra computadora que haga algo como si estuviera al lado. El nodo atiende ese tipo de peticiones.

Hay varias formas de conversar:



Forma	Qué aspecto tiene	Dónde la encontrarás
JSON-RPC	Una sola dirección; en la solicitud se nombran el método y los parámetros	Ethereum y similares, Solana, Bitcoin
REST	Cada pregunta tiene su propia ruta en la dirección	TRON, TON, Aptos
gRPC	Mensajes binarios compactos con un esquema estricto	Intercambio entre programas, por ejemplo en Sui
WebSocket	Una conexión que no se cierra	Cuando el nodo debe avisar de las novedades por sí mismo

WebSocket no es un idioma, sino una **línea de comunicación**: por ella viajan en Ethereum los mismos mensajes JSON-RPC. Algunas redes tienen varias formas a la vez.

JSON-RPC pieza por pieza



JSON es una forma sencilla de escribir datos: llaves, nombres entre comillas y valores. Este es el comando del capítulo 3, desglosado línea por línea:

Trozo del comando	Qué significa
`curl`	programa que envía una solicitud desde la terminal
`https://…`	URL del endpoint
`-H '…'`	la nota «dentro va JSON»
`-d '{…}'`	la pregunta en sí

En la respuesta se miran `result` (lo que salió) y `error` (lo que no salió). Unos nodos envían solo uno de estos campos; otros, los dos, y entonces el vacío vale `null`. El `id` de la respuesta es el mismo de la pregunta: así se emparejan.

Varias preguntas de una vez

Las preguntas se pueden juntar en una lista entre corchetes. Esto se llama **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"}]
```

La primera respuesta es el número de bloque, 26 502 176; la segunda, el número de la red: el de Ethereum es 1. Según las reglas de JSON-RPC, las respuestas de un lote pueden llegar en cualquier orden, así que fíjate en el `id`. Y recuerda: el lote ahorra viajes, pero no límite. En RpcNode un lote de diez preguntas cuenta como diez solicitudes, y se acepta o se rechaza entero.

Y ahora, REST

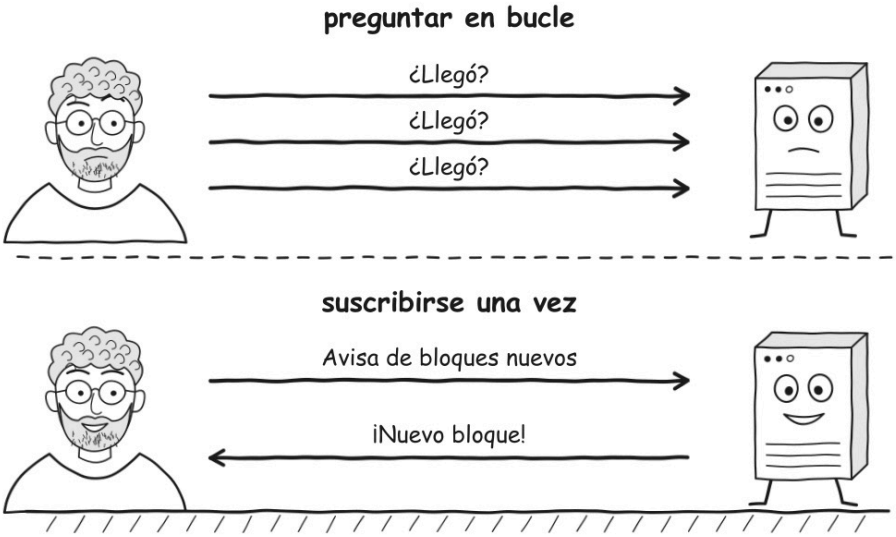
Aquí la pregunta va incrustada en la propia dirección. Pidamos a la red TRON el bloque más reciente:

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

Llega una larga respuesta sobre un bloque con transferencias. Cuando se escribía este capítulo era el bloque número 88 351 889, con 194 transacciones. Otra pregunta, otra ruta detrás de la dirección.

WebSocket: no preguntar, sino escuchar

Preguntar «¿ya llegó?» cada segundo es un derroche. Por una línea abierta te puedes suscribir una sola vez y el nodo mismo te irá mandando las novedades.



Los nodos públicos de RpcNode no responden por WebSocket; hace falta un endpoint personal. Su dirección empieza por `wss://`. Este es un esbozo en JavaScript:

```
const ws = new WebSocket('wss://TU_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('se cortó la línea')
```

Las líneas se cortan. El programa debe saber reconectarse, suscribirse otra vez y comprobar qué se perdió. Las suscripciones también gastan créditos.

gRPC

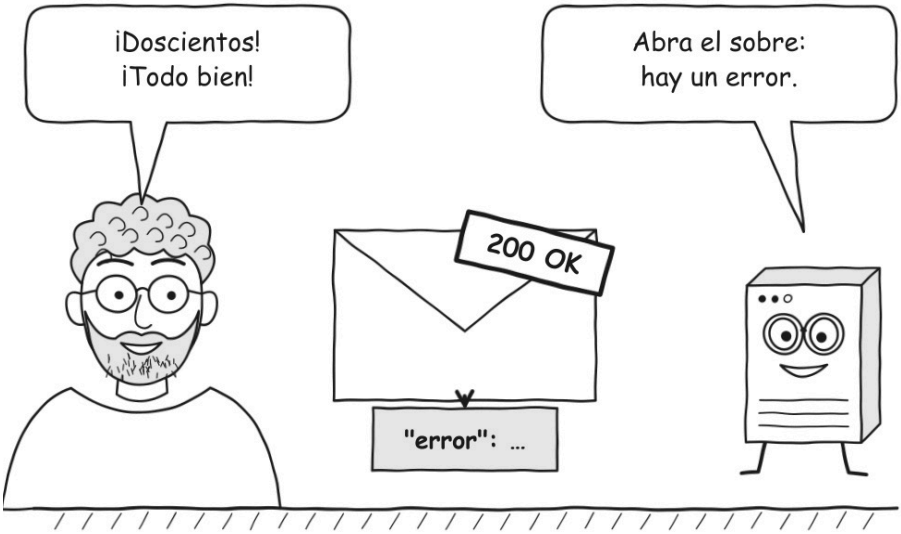
Es una forma pensada para programas a los que les importan la velocidad y un formato estricto. Los mensajes no se leen a simple vista y sus campos se describen de antemano en un archivo aparte. Una tienda que empieza normalmente no lo necesita. En RpcNode, gRPC no existe para todas las redes: mira la lista de transportes del endpoint.

La trampa: «200 significa que todo va bien»

200 es el código de entrega: el sobre llegó. Pero dentro puede haber un error. Pidamos el saldo de la dirección «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"]}'
```

El código de respuesta es 200 y dentro hay un `error` con el número `-32602` : parámetros no válidos.



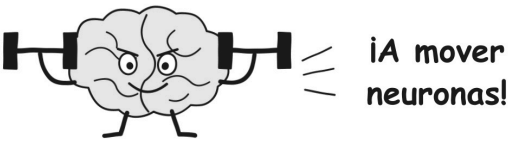
Por eso el programa comprueba siempre el código y también el campo `error`.

Errores frecuentes

Código	Qué pasó	Qué hacer
`-32700`	No se puede interpretar el texto de la solicitud	Revisar llaves y comillas
`-32600`	La solicitud no respeta las reglas	Revisar los campos
`-32601`	Método no encontrado o no disponible	Revisar el nombre y el plan
`-32602`	Parámetros no válidos	Revisar los valores
`-32005` (HTTP 429)	RpcNode: demasiado seguido	Esperar y preguntar con menos frecuencia
`-32006` (HTTP 402)	RpcNode: no alcanzan los créditos	Revisar el consumo y el plan

Los cuatro primeros códigos son del estándar JSON-RPC 2.0; los dos últimos, de RpcNode.

Para pensar



1. En la primera solicitud del lote escribe `eth_blockNumbr` en lugar de `eth_blockNumber`. ¿Qué código llegará? ¿Contestarán a la segunda pregunta, que está bien?
2. ¿Qué es mejor para un marcador de «último bloque» que debe actualizarse solo: preguntar cada segundo o suscribirse?
3. El programa recibió HTTP 200 y marcó enseguida el pedido como pagado. ¿Qué está mal?

Respuestas: 1) `-32601`, y uno solo para todos: RpcNode rechaza el lote entero y la segunda pregunta también se queda sin respuesta. 2) Suscribirse. 3) No se comprobaron ni el campo `error` ni la propia transferencia (capítulo 7).

Lo esencial

- JSON-RPC nombra el método en la solicitud; REST, en la dirección.
- WebSocket mantiene la línea abierta para que el nodo avise de las novedades.
- El éxito no es el código 200, sino una respuesta sin `error` y con lo que esperabas.

A continuación: capítulo 12. Miremos al otro lado de la ventanilla: cómo vive un nodo propio, qué es la sincronización y por qué se rompe.

Capítulo 12. Un nodo propio por dentro: instalación, sincronización y averías

«¡Si eso es cosa de una tarde!»

Un programador conocido intenta convencer a Artyom: «Lo descargas, lo arrancas y ya tienes tu propio nodo». Arrancarlo, es verdad, no lleva mucho. Lo largo es lo que empieza después.

Entendamos: la vida de un nodo

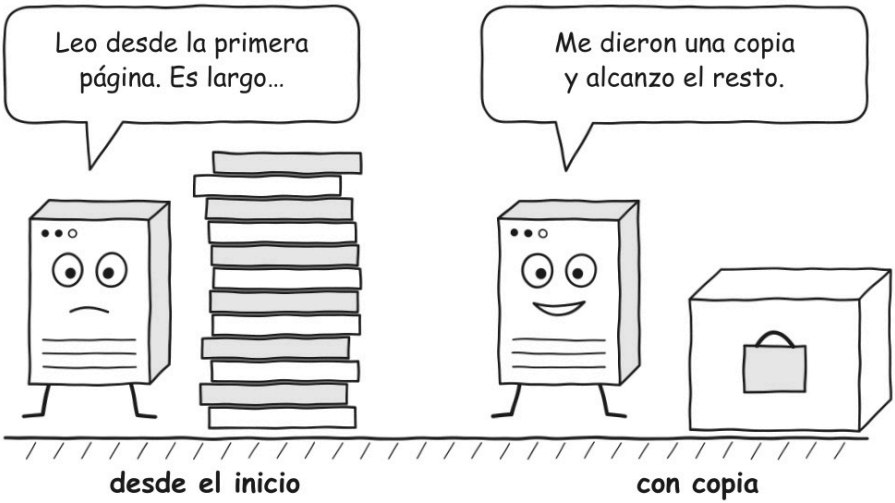


4 y 5 se repiten mientras el nodo viva

1. **Servidor.** Los requisitos los vimos en el capítulo 2.
2. **Programa.** Cada red tiene el suyo, y a menudo varios para elegir. En Ethereum hacen falta dos a la vez: el **cliente de ejecución** (ejecuta las transferencias y los contratos) y el **cliente de consenso** (vigila qué cadena es la buena).
3. **Sincronización.** El nodo alcanza a la red.
4. **Vigilancia.** Si no se ha atrasado, si queda espacio.
5. **Actualizaciones.** La red cambia sus reglas y el programa se cambia de antemano.

Sincronización: alcanzar a la red

Un nodo nuevo no sabe nada. Mientras no haya alcanzado a los demás, es inútil preguntarle por un saldo: contestará con el pasado.



- **Desde el principio.** El nodo comprueba cada bloque por sí mismo. Fiable y lento. La primera descarga de Bitcoin son unos 740 GB.
- **Desde una instantánea.** Se toma una copia ya hecha de la base de datos y se alcanza el resto. Es más rápido, pero confías en quien hizo la instantánea.
- **Formas intermedias.** Los programas de Ethereum suelen descargar el estado reciente y cotejarlo con las cabeceras de los bloques.

En tiempo, va de varias horas a muchos días: depende de la red, del método, del disco y de la conexión.

Probemos: una configuración de dos líneas

Así puede empezar el archivo de configuración `bitcoin.conf`:

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

Línea	Qué significa
`server=1`	aceptar preguntas por RPC; mientras no se permita otra cosa, solo desde la misma computadora
`prune=5000`	nodo podado (capítulo 10): borrar los bloques antiguos y conservar de ellos unos 5 GB; los datos auxiliares ocupan además su espacio

No son unas instrucciones completas. Antes de instalar, lee la documentación de tu programa.

Cómo saber que el nodo ya alcanzó a la red

Al nodo se le puede preguntar directamente. En Ethereum:

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

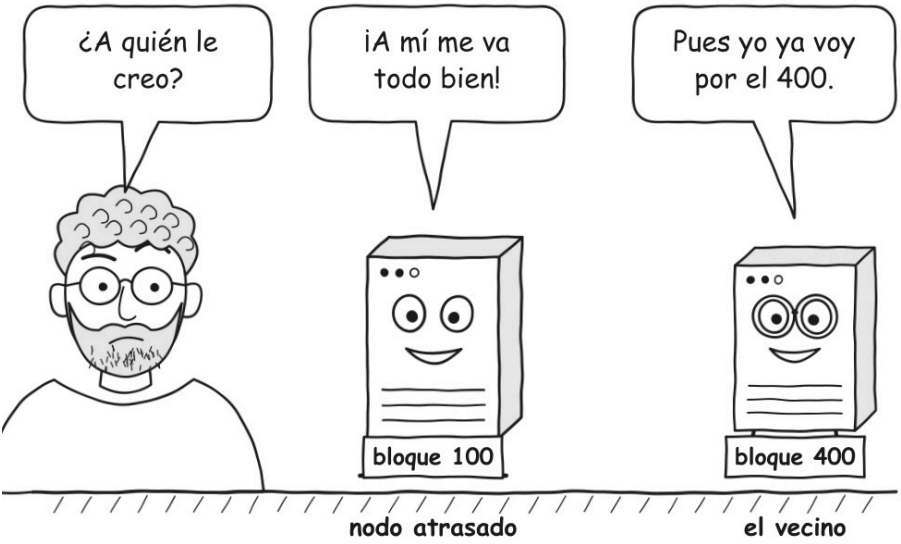
```
{"jsonrpc":"2.0","id":1,"result":false}
```

`false` significa «ahora no estoy poniéndome al día». Mientras el nodo se esté poniendo al día, en su lugar llegan números: dónde está y hacia dónde va. En Bitcoin se mira la respuesta de `getblockchaininfo` del capítulo 10:

Campo	Valor al comprobar	Qué significa
`blocks`	977 715	cuántos bloques se han verificado
`headers`	977 715	de cuántos bloques se tiene noticia
`initialblockdownload`	`false`	la primera descarga quedó atrás

La trampa: «él mismo dijo que todo iba bien»

El nodo juzga por lo que sabe. Si ha perdido el contacto con los demás, puede que ni sospeche que se ha atrasado.



Por eso se comprueban dos cosas: la **frescura del último bloque** y la **comparación con otra fuente**. La frescura es fácil de saber. Pide la hora del último bloque:

```
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"
```

Son los segundos transcurridos desde 1970. Réstalos de la hora actual (pon tu propio número):

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

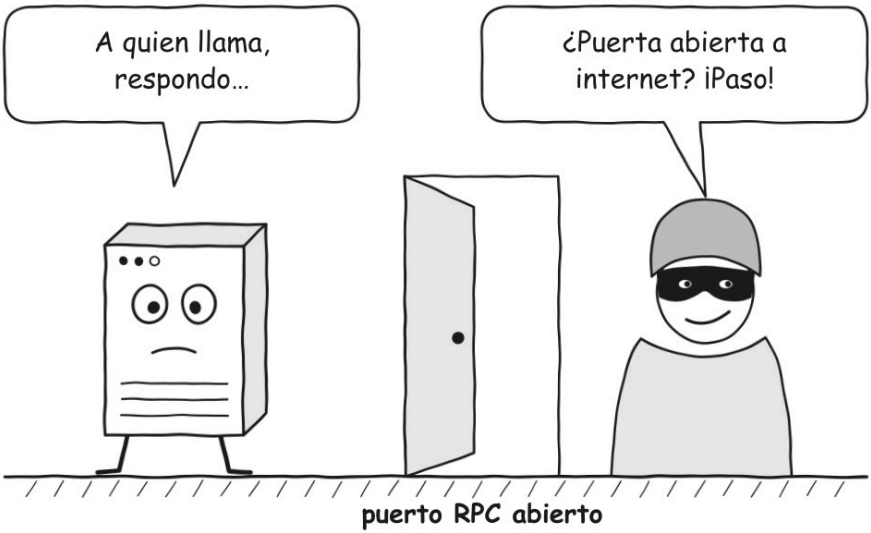
Sale la edad del bloque en segundos. Cuando se escribía este capítulo era de 7 segundos: para Ethereum, con un bloque cada 12 segundos, es lo normal. Y si en tu nodo son miles, se ha atrasado. ¿Recuerdas la pregunta del capítulo 1 sobre el nodo con una hora de atraso?

Qué se rompe

Qué se ve	Causa frecuente	Qué hacer
El nodo se detuvo	Se acabó el espacio en disco	Vigilar el espacio y ampliarlo de antemano
Siempre va atrasado	Disco o conexión flojos, pocos vecinos	Revisar el hardware y la red
Se «atascó» tras una actualización de la red	Versión antigua del programa	Actualizar antes de la fecha anunciada
No arranca tras un corte de luz	Se dañó la base de datos	Restaurar desde una copia o sincronizar de nuevo
Va lento sin motivo aparente	Lo cargan solicitudes ajenas	Cerrar el acceso a los extraños

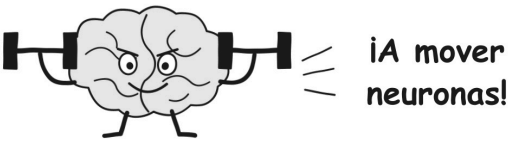
Segunda trampa: la puerta abierta

Bitcoin Core y Geth, de partida, aceptan preguntas por RPC solo desde su propia computadora. Y no es por casualidad. Si abres el puerto a todo internet, dejas entrar a cualquiera, y si en el nodo hay una billetera, arriesgas dinero.



- Permite el acceso solo a tus propias direcciones (cortafuegos).
- Pon delante del nodo una comprobación de clave.
- No guardes una billetera con dinero en el nodo que atiende al sitio web.

Para pensar



1. `eth_syncing` responde `false`, pero el último bloque tiene 40 minutos. ¿Está sano el nodo?
2. El disco es de 2000 GB, hay 1900 GB ocupados y cada día se suman 5 GB. ¿En cuántos días se parará todo?
3. El sitio está en otro servidor. ¿Se puede abrir el puerto RPC a todos «para que el sitio pueda preguntar»?

Respuestas: 1) No. 2) En 20 días como mucho, y hay que empezar a arreglarlo hoy. 3) No: autoriza solo la dirección del sitio.

Lo esencial

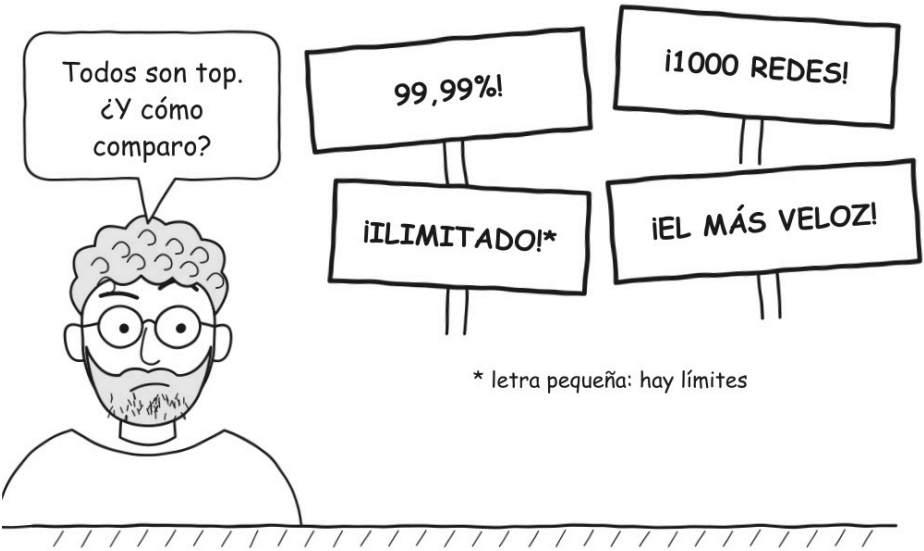
- Arrancar un nodo es el principio del trabajo, no el final.
- La salud se comprueba por la frescura del bloque y comparando con otra fuente.
- El puerto RPC no se abre a todo internet.

A continuación: capítulo 13. Si nada de esto es tu pasatiempo favorito: cómo elegir proveedor y qué preguntarle.

Capítulo 13. Cómo elegir proveedor

Todos prometen lo mejor

Artyom lo ha decidido: un nodo propio no es para él. Abre los sitios de los proveedores y solo encuentra signos de exclamación.



Entendamos: primero, tú

Comparar proveedores sin conocer tus necesidades no sirve de nada. Primero responde:

- ¿Qué redes necesitas? ¿Hacen falta las de prueba?
- ¿Cuántas solicitudes por segundo y al mes?
- ¿Necesitas la historia antigua (archivo)?
- ¿Necesitas enterarte de los eventos al instante (WebSocket, notificaciones)?

Cuenta tus solicitudes

Artyom tiene 20 pedidos esperando el pago y el sitio comprueba cada uno cada 10 segundos:

> 20 pedidos × 6 veces por minuto = 120 solicitudes por minuto, es decir, 2 por segundo.
> En 30 días: 5 184 000 solicitudes.

Después multiplica por el precio del método en créditos: cada método tiene el suyo. Y ahora piensa si de verdad hace falta preguntar tan seguido. Cada 30 segundos, y las solicitudes se reducen a la tercera parte.

La lista de preguntas



Pregunta	Mala señal
¿Están **mis redes**? ¿Y en ellas los métodos que necesito?	Una larga lista de redes sin lista de métodos
Límites: solicitudes por segundo y en total; ¿qué pasa si se superan?	«Ilimitado» sin cifras; recargos poco claros
Archivo, WebSocket, gRPC: ¿en qué plan y para qué redes?	«Lo tenemos todo», pero sin detalles
Fallo de un nodo: ¿te pasan a otro? ¿Dónde están los servidores?	No hay respuesta
Disponibilidad: ¿qué prometen, cómo la miden y qué pasa si no se cumple?	Un porcentaje sin periodo ni condiciones
Protección: ¿se pueden restringir direcciones y sitios? ¿Qué registran de las solicitudes?	La clave le funciona a cualquiera
Soporte: ¿dónde se escribe y en qué idioma contestan?	Solo un formulario anónimo
Salida: ¿son métodos estándar? ¿Se puede pagar mes a mes?	Un largo plazo obligatorio

Lo que esconde un porcentaje

«Disponibilidad del 99%» suena casi como «siempre». Calculemos cuánta caída permite eso en 30 días:

Prometido	Puede no funcionar
99%	7 horas y 12 minutos
99,9%	unos 43 minutos
99,99%	unos 4 minutos

Y pregunta **qué exactamente** se considera que funciona: el sitio del proveedor o tu endpoint en tu red.

Probemos: mide la velocidad por tu cuenta



Las mediciones ajenas se hicieron desde un lugar ajeno. Lanza cinco solicitudes desde el servidor donde va a vivir tu sitio:

```
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
```

Verás cinco números: el tiempo de respuesta en segundos. No mires solo el mejor: al cliente también le tocará el peor. Aquí la conexión se abre de nuevo cada vez; un programa con conexión permanente suele ser más rápido. Y no compares solo la velocidad, sino también la frescura del bloque (capítulo 12).

Hazlas también estas preguntas a nosotros

El libro trata de RpcNode, así que vamos a examinarlo también. Datos a 27 de noviembre de 2026.

Pregunta	RpcNode
Redes	25 en la lista del sitio
Probar gratis	plan Free y nodos públicos sin clave
Límites	Free: 15 solicitudes por segundo y 15 millones de créditos; Starter: 40 y 50 millones
Al superarlos	no hay recargo por exceso: las solicitudes se rechazan (429 o 402, capítulo 11)
Archivo	en los planes de pago
WebSocket y gRPC	mediante un endpoint personal, no en todas las redes
Dónde están los servidores	Fráncfort, Estados Unidos, Japón
Protección	el endpoint se puede restringir por direcciones IP y por sitios
Privacidad	el proveedor ve tus solicitudes (capítulo 3)
Disponibilidad y soporte	consulta las condiciones vigentes en rpcnode.dev

La trampa: elegir por la cifra más grande

Mil redes no ayudan si las dos tuyas funcionan mal. El precio más bajo no ayuda si el método que necesitas es caro o no está disponible. Toma el plan gratuito o el más barato y pásale durante varios días tus solicitudes reales.

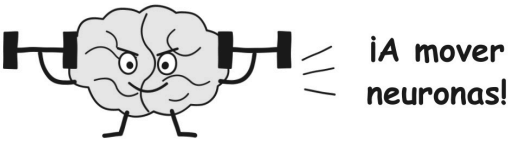
No eches raíces



- Guarda la URL del endpoint en un solo ajuste, no dentro del código.
- En lo posible, usa los métodos estándar de la red.

- Prepara una alternativa de reserva y algún día cámbiate a ella para probarla.

Para pensar



1. 20 pedidos, comprobación cada 30 segundos. ¿Cuántas solicitudes son en 30 días?
2. Se prometió un 99,9% y el endpoint estuvo dos horas sin funcionar en el mes. ¿Se cumplió la promesa?
3. La URL del endpoint está escrita en 14 lugares del código. ¿Qué tiene de malo?

Respuestas: 1) 1 728 000. 2) No: se permiten unos 43 minutos. 3) Al cambiarla es fácil olvidar un sitio, y una clave dentro del código se filtra con más facilidad.

Lo esencial

- Primero calcula tus necesidades y después compara.
- Comprueba las promesas con tus mediciones y tus solicitudes.
- Déjate abierta la posibilidad de irte.

Fin de la tercera parte. Artyom ya sabe qué es un nodo, cómo hablarle y a quién pedírselo. ¡Pero las redes son distintas! **A continuación:** parte 4, capítulo 14. Bitcoin y sus parientes.

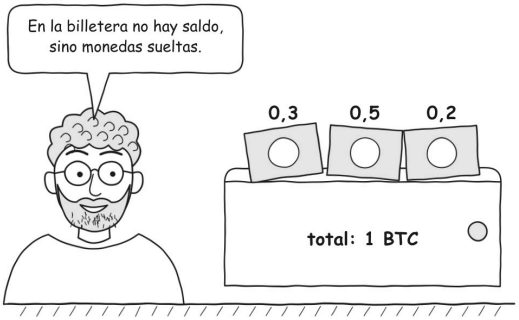
Capítulo 14. Bitcoin y sus parientes

«¿Y dónde tiene el saldo?»

Un cliente quiere pagar con bitcoins. Artyom ya tiene experiencia: ahora mismo le pide al nodo el saldo de la dirección, como en Ethereum. Pero el nodo de Bitcoin no tiene una pregunta tan sencilla. ¿Por qué?

Entendamos: no es una cuenta, son monedas

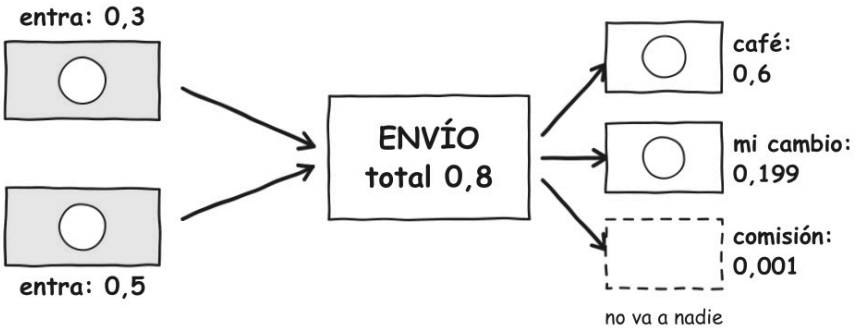
En Ethereum una dirección tiene un saldo, como una cuenta bancaria. En Bitcoin, no. Allí hay «trozos de dinero» separados, como billetes de distinto valor.



Cada uno de esos trozos se llama **UTXO** (salida no gastada). El saldo es la suma de todos tus trozos sin gastar. Lo calcula la billetera o un indexador (capítulo 10).

La transferencia: entradas, salidas y cambio

Un billete no se puede rasgar por la mitad. Hay que entregarlo entero y recibir el cambio. Aquí es igual: la transferencia gasta por completo trozos antiguos (**entradas**) y crea otros nuevos (**salidas**).



- Una salida es para el destinatario.
- La segunda es el **cambio**, para uno mismo.
- Todo lo que no se anota en ninguna salida se lo queda el minero. Es la **comisión**: entradas menos salidas.

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

El precio de la comisión no se calcula según la cantidad, sino según el tamaño de la transferencia: en **satoshis por byte virtual**. Un satoshi es la cienmillonésima parte de un bitcoin. Cuantos más trozos entran, más grande es la transferencia y más cara sale.

La primerísima transferencia entre personas

El 12 de enero de 2009, en el bloque 170, un trozo de 50 BTC se convirtió en dos: 10 BTC y 40 BTC. El destinatario de los diez, el programador Hal Finney, contó él mismo que Satoshi se los envió para probar. El cambio, 40 BTC, volvió a quien envió. No hubo comisión.

En la blockchain no hay nombres: lo de Finney se sabe por su propio relato (bitcointalk.org, año 2013).

Esa transferencia todavía se puede consultar:

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

En la respuesta busca `vin` (las entradas, aquí una) y `vout` (las salidas, aquí dos: `10.0` y `40.0`).

Probemos: ¿está gastada la moneda?

El método `gettxout` pregunta: «¿sigue entero este trozo?». El trozo se identifica por el número de la transferencia y el número de la salida, contando desde cero. Preguntemos por aquellos 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` : no existe ese trozo sin gastar; se gastó hace mucho. Y ahora, la recompensa del primerísimo bloque después del inicial:

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

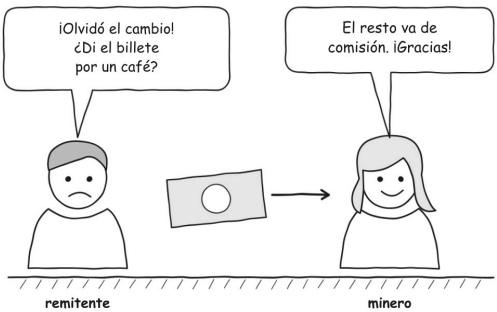
Aquí llegan `"value":50.00000000` y casi un millón de confirmaciones. Cuando se escribía este capítulo, nadie había tocado todavía esos 50 BTC.

Tres datos más sobre Bitcoin

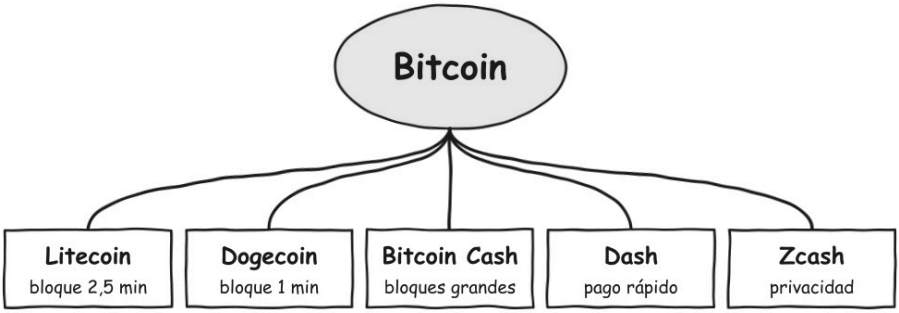
- En total no habrá más de **21 millones** de monedas.
- La recompensa del minero no se puede gastar hasta que haya 100 bloques encima.
- Para sumas grandes se esperan 6 confirmaciones, cerca de una hora (capítulo 8).

La trampa: el cambio olvidado

Una billetera normal añade el cambio sola. Pero si un programa arma la transferencia a mano y se olvida la salida del cambio, todo el resto se convierte en comisión.



Los parientes



se parecen, pero cada uno tiene su red y sus monedas

Red	Un bloque, aprox.	En qué se distingue	La dirección empieza por
Bitcoin	10 min	el mayor de la familia	`1`, `3`, `bc1`
Litecoin	2,5 min	otro algoritmo de minería, Scrypt	`L`, `M`, `ltc1`
Dogecoin	1 min	la emisión no tiene tope; se puede minar a la vez que Litecoin	`D`
Bitcoin Cash	10 min	bloques mucho más grandes	`bitcoincash:q…`
Dash	2,5 min	pagos rápidos (InstantSend) y protección contra la sustitución de bloques (ChainLocks)	`X`
Zcash	75 s	las transferencias se pueden ocultar	visibles `t1`; ocultas `zs`, `u1`

Están construidos de forma parecida, así que las preguntas también son casi iguales. Preguntemos de una vez la altura de la cadena a cinco redes:

```
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
```

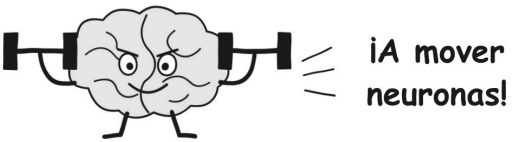
```
bitcoin: {"jsonrpc":"2.0","result":977717,"id":1}
litecoin: {"result":3219941,"error":null,"id":1}
doge: {"result":6474162,"error":null,"id":1}
bch: {"result":979125,"error":null,"id":1}
dash: {"result":2578936,"error":null,"id":1}
```

Segunda trampa: «si hay error, es que algo falló»

Mira las respuestas: en cuatro redes el campo `error` está siempre. El valor `null` significa «todo va bien». No compruebes si el campo existe, sino lo que contiene, tal como aconsejaba el capítulo 11.

Y no confundas las redes. Los parientes no aceptan las monedas de los otros. El formato antiguo de dirección de Bitcoin Cash tenía el mismo aspecto que el de Bitcoin; por eso allí introdujeron uno nuevo, con el prefijo `bitcoincash:`.

Para pensar



1. En la billetera hay trozos de 0,3, 0,5 y 0,2 BTC. Hay que pagar 0,4 y la comisión es 0,001. ¿Basta con un trozo y cuánto será el cambio?
2. ¿Por qué enviar 1 BTC reunido a partir de cien trozos pequeños sale más caro que con uno entero?
3. La entrada es de 1,0 BTC y la única salida, de 0,4 BTC. ¿Adónde fueron los 0,6?

Respuestas: 1) sí, el trozo de 0,5; el cambio es 0,099. 2) Más entradas, mayor tamaño de la transferencia. 3) Es la comisión. Lo más probable es que olvidaran el cambio.

Lo esencial

- En Bitcoin no hay saldo de cuenta, hay trozos sin gastar (UTXO).
- La comisión son las entradas menos las salidas; no olvides el cambio.
- Los parientes se parecen en su construcción, pero son redes distintas.

A continuación: capítulo 15. Ethereum: cuentas, gas y eventos.

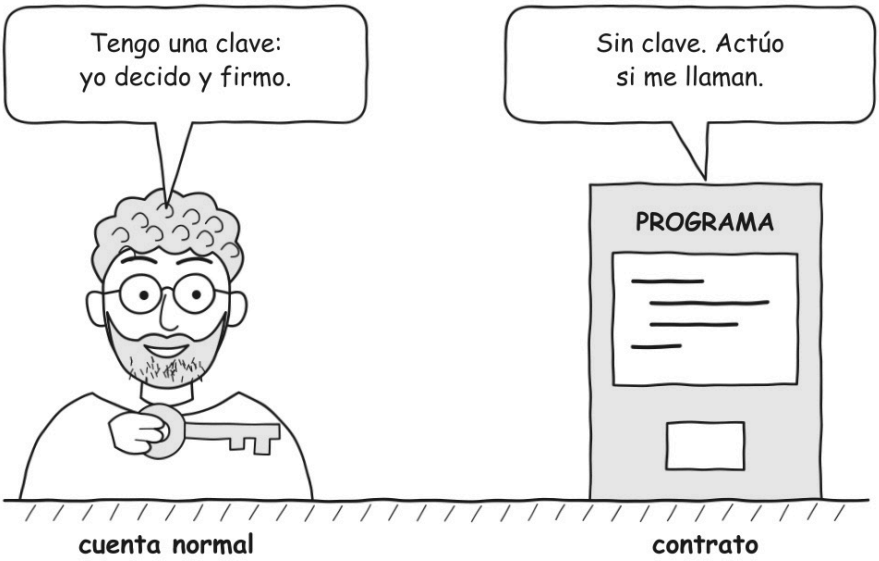
Capítulo 15. Ethereum: cuentas, gas y eventos

«¿Por qué enviar dólares sale más caro?»

Artyom le envió a su proveedor de café un poco de ETH y después unos USDT. Las cantidades eran parecidas, pero por los USDT le cobraron más del doble de «gas». ¿Qué es ese gas y por qué pasa eso?

Entendamos: dos tipos de cuentas

En Ethereum una dirección tiene un saldo, como en un banco. No como los trozos de Bitcoin (capítulo 14). Y las cuentas son de dos tipos.



	Cuenta normal	Contrato
Quién la maneja	el dueño de la clave	su programa
Puede iniciar una transferencia por sí misma	sí	no, solo cuando lo llaman
Tiene código	normalmente no	sí

Los programas los ejecuta la **EVM**, la máquina virtual de Ethereum. Es como una consola de juegos dentro de cada nodo: el mismo programa da el mismo resultado en todas partes.

Probemos: dónde está el programa

El método `eth_getCode` muestra el código que hay en una dirección. Preguntemos por el contrato de USDT y quedémonos solo con el principio de la respuesta:

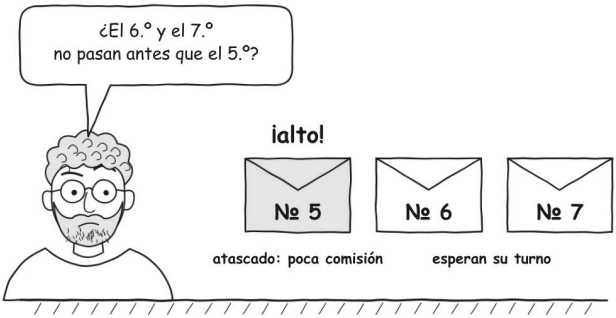
```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getCode","params":
  ["0xdac17f958d2ee523a2206206994597c13d831ec7","latest"]}' | head -c 80; echo
```

Verás el comienzo de una larga cadena de caracteres: ese es el programa, más de 11 mil bytes. Cambia la dirección por `0x0000000000000000000000000000000000000dEaD` y llegará `"0x"`, es decir, no hay código.

Desde 2025, una cuenta normal también puede encargar sus acciones a un programa (actualización Pectra). En ese caso la respuesta empieza por `0xef0100`. Así que «tiene código» ya no siempre significa «es un contrato».

El número de orden: nonce

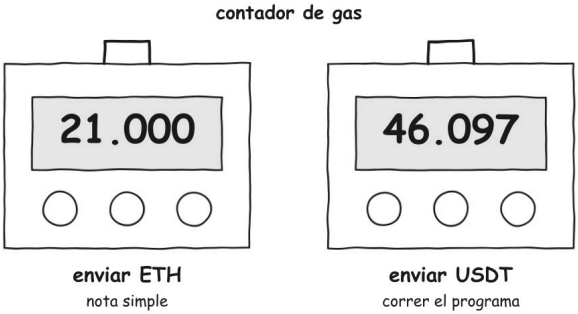
Cada transacción de una cuenta recibe un número: 0, 1, 2, etcétera. Se llama **nonce**. La red las ejecuta estrictamente en orden y no aceptará el mismo número por segunda vez.



Si una transferencia con una comisión pequeña se atasca, las siguientes esperan detrás. Las billeteras ofrecen «acelerar»: envían una transferencia nueva con el mismo número y una comisión mayor.

Gas: el contador de trabajo

El **gas** mide cuánto trabajo exige una transacción. La simple anotación «transferir ETH» cuesta hoy 21 000. Transferir un token es ejecutar un programa, y se necesita más gas.



Comisión = gas gastado × (precio base + propina)

- El **precio base** lo fija la propia red. Ese dinero **se quema**.
- La **propina** se la queda el validador: cuanto más alta, con más ganas te recogen.
- El precio del gas se mide en **gwei**, la milmillonésima parte de un ETH.

En un bloque cabe una cantidad limitada de gas. Cuando los bloques van llenos a más de la mitad, el precio base sube; cuando van a menos, baja. Por ejemplo, en el bloque 26 502 250 cabían 60 millones de gas y estaba lleno al 93%.

La billetera fija un **límite de gas**. Solo se cobra lo gastado. Pero si el límite no alcanza, la transferencia no se ejecuta y el gas gastado no se devuelve.

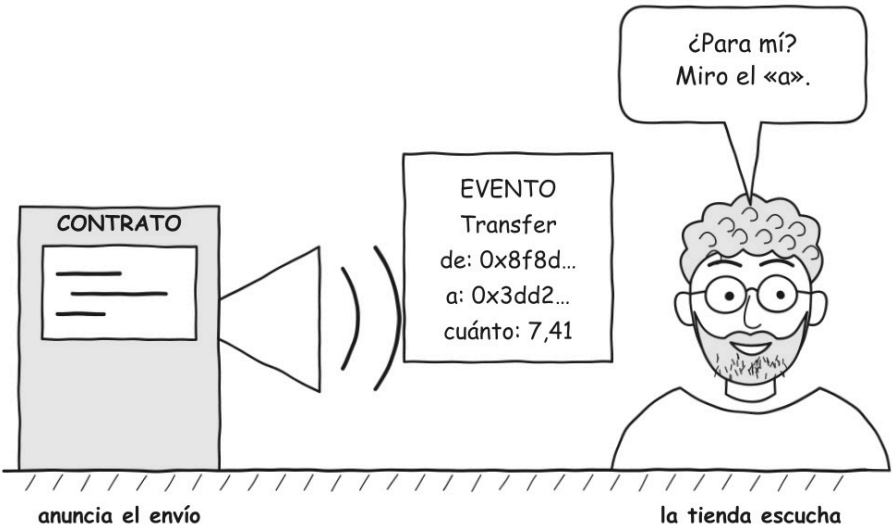
Una transferencia de USDT de verdad

Es una transferencia de 7,41 USDT de ese mismo bloque. Quien envió ya había hecho para entonces 423 464 transacciones.

Qué	Valor
gas gastado	46 097
precio base	0,058 gwei
propina	2 gwei
comisión	unos 0,000095 ETH
de ellos se quemaron	unos 0,000003 ETH

Eventos: cómo informa un contrato de una transferencia

Los saldos de los tokens están dentro del contrato. Para que el mundo se entere de una transferencia, el contrato anuncia un **evento** (log). En los tokens se llama `Transfer`: de quién, a quién y cuánto.



Los eventos están en el recibo de la transacción del capítulo 7. Pidamos el recibo de esta transferencia:

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

Busca `logs` en la respuesta:

Campo	Principio y final	Qué significa
`address`	`0xdac1…1ec7`	quién lo anunció: el contrato de USDT
`topics[0]`	`0xddf2…b3ef`	tipo de evento: Transfer
`topics[1]`	`…8f8d…884e`	de quién
`topics[2]`	`…3dd2…a9c0`	a quién
`data`	`…711150`	7 410 000, es decir, 7,41 USDT

Y el método `eth_getLogs` sabe buscar eventos. Contemos las transferencias de USDT del bloque más reciente:

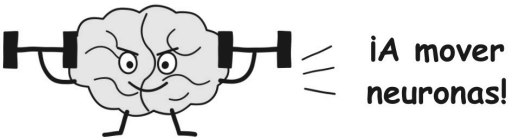
```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getLogs","params":[{
  "address":"0xdac17f958d2ee523a2206206994597c13d831ec7",
  "topics":["0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef"],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

Sale el número de transferencias de un bloque. Sobre bloques pasados el nodo público no contestará: eso ya es archivo (capítulo 10).

La trampa: «solo escucho eventos»

- **La transferencia del propio ETH no crea un evento Transfer.** Una tienda que solo espera eventos se saltará ese pago. Futuras actualizaciones de la red podrían cambiar esto.
- **El ETH puede llegar desde un contrato**, por ejemplo desde algunas billeteras. Entonces en la propia transacción no se ve tu dirección.
- **Cualquier contrato puede anunciar un evento.** Comprueba `address`: si es el token auténtico (capítulo 9).

Para pensar



1. Transferencia de ETH, 21 000 de gas, precio con propina incluida de 2 gwei. ¿Cuál es la comisión en gwei y en ETH?
2. Varios bloques seguidos van llenos al 30%. ¿Qué pasará con el precio base?
3. La tienda solo vigila los eventos Transfer del contrato de USDT. El cliente pagó en ETH. ¿Lo verá la tienda?

Respuestas: 1) 42 000 gwei, es decir, 0,000042 ETH. 2) Irá bajando. 3) No.

Lo esencial

- Las cuentas las maneja una clave o un programa.
- La comisión es el gas multiplicado por el precio: el base y la propina.
- De las transferencias de tokens te enteras por los eventos, y el ETH hay que capturarlo aparte.

A continuación: capítulo 16. Ethereum tiene mucha familia: otras redes EVM y los «segundos pisos».

Capítulo 16. La familia de Ethereum y los «segundos pisos»

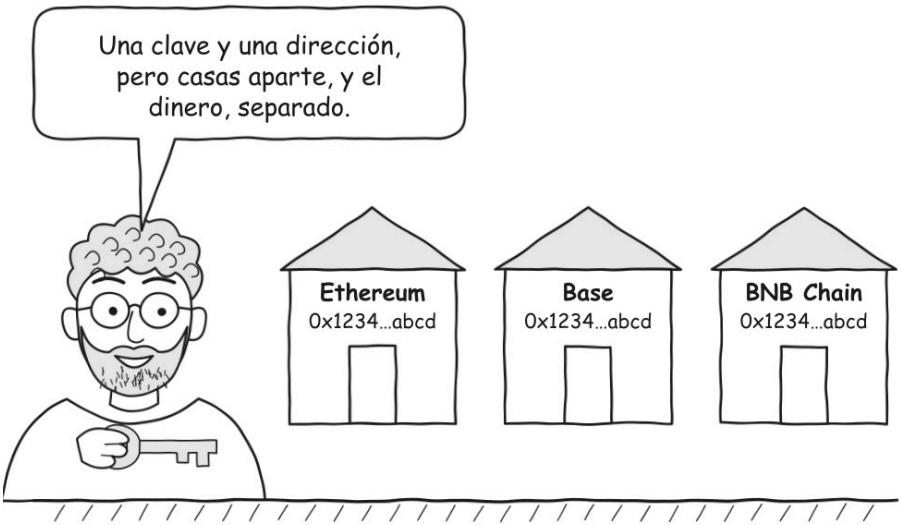
«Lo envié por Base»

Un cliente escribe: «Envié USDC por Base, la dirección es la misma». Artyom está desconcertado. ¿Es otro token? ¿Otra red? ¿Y por qué la dirección es la misma?

Entendamos: la misma consola, casas distintas

Muchas redes adoptaron la EVM del capítulo 15. Por eso tienen las mismas direcciones y los mismos métodos `eth_…`, y un mismo programa de contrato puede funcionar en cada una. Esas redes se llaman **compatibles con EVM**.

Pero sus cuadernos son **distintos**. Bloques propios, saldos propios, comisiones propias.



Para no confundir las redes, cada una tiene un número: el **chain ID**. Forma parte de la firma, así que una transferencia firmada para una red no la acepta otra.

Probemos: preguntar su 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
```

Llegarán números en hexadecimal. Aquí están en números corrientes, junto con la frecuencia con que salían bloques al comprobarlo:

Red	Chain ID	Un bloque aprox. cada	Con qué se paga la comisión
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

Parientes del primer tipo: los independientes

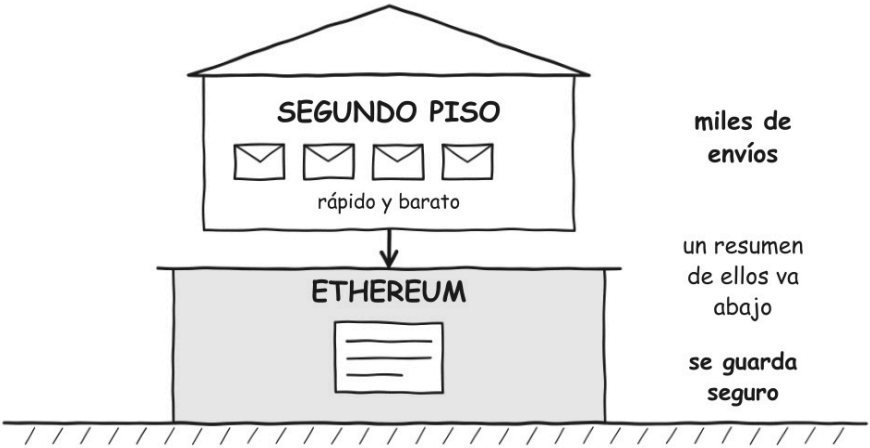
Tienen sus propios validadores y se ponen de acuerdo ellos mismos sobre los bloques.



Polygon PoS, además, deja periódicamente en Ethereum marcas sobre sus bloques. Entre los independientes está también Monad, puesta en marcha en noviembre de 2025.

Parientes del segundo tipo: los segundos pisos (L2)

Una **L2** ejecuta las transferencias en su propia casa, rápido y barato, y envía a Ethereum las anotaciones comprimidas sobre ellas. El tipo más frecuente se llama **rollup**, «enrollado».



- El orden de las transferencias lo suele fijar un único operador, el **secuenciador**. Por eso todo es tan rápido. Y por eso en ese punto confías en él.
- Los rollups **optimistas** (Arbitrum One, OP Mainnet) dan las anotaciones por válidas, pero dejan tiempo para impugnarlas.
- Los rollups **ZK** (por ejemplo, Mantle) adjuntan una prueba matemática.
- Base pasó en 2026 a su propio código y combina varios tipos de pruebas.

Por el tiempo que lleva la comprobación, no podrás devolver rápido el dinero a Ethereum por el puente oficial:

Desde la red	Cuánto esperar para retirar a Ethereum
Arbitrum One	no menos de 7 días
OP Mainnet	7 días
Base	5 días; en casos concretos, 1 día

Según la documentación de las redes a 27 de noviembre de 2026. Estos plazos cambian: en OP Mainnet hay una reforma prevista para enero de 2027.

«Lo vi enseguida» no significa «definitivo»

¿Recuerdas `latest`, `safe` y `finalized` del capítulo 8? Esto contestó el nodo de Base:

Qué se preguntó	Atraso respecto al bloque más reciente
`latest`	—
`safe`	unos 17 segundos
`finalized`	unos 19 minutos

El bloque aparece en fracciones de segundo, pero se vuelve definitivo cuando su anotación ya es definitiva en el propio Ethereum.

La comisión en el segundo piso

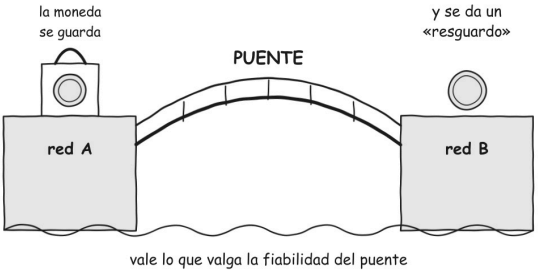
Se compone principalmente de dos partes: el trabajo en la propia L2 y el pago por escribir en Ethereum. A veces hay también pequeños añadidos. En un recibo de Base, la segunda parte está en el campo `l1Fee`:

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

Aquí por el trabajo en Base cobraron unos 0,0000035 ETH, y por escribir en Ethereum, unos 0,0000000006 ETH. No siempre sale tan barato.

Puentes

Para «mudarse» de una red a otra se usa un **puente**. A menudo encierra las monedas en una red y emite en la otra un sustituto de ellas.



Por eso **USDC** y **USDC.e** son tokens distintos, con direcciones de contrato distintas. El primero lo emite la propia empresa Circle; el segundo aparece a través de un puente.

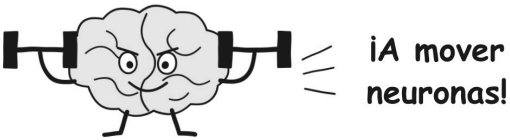
La trampa: lo envié por la red equivocada

En el capítulo 6 dijimos que la tienda no verá ese pago. Es verdad. Pero ¿se perdió el dinero?

- **La clave de la dirección la tienes tú:** normalmente el dinero se puede rescatar cambiando la billetera a esa red. Para reenviarlo necesitarás la moneda de esa red para la comisión.
- **Es una dirección de un exchange:** no tienes la clave. Si el exchange no admite esa red, lo más probable es que el dinero no se pueda recuperar.
- **Es una billetera-contrato:** en otra red puede que no exista. Hace falta la ayuda de un experto.

> **Regla:** no le des a nadie tu frase secreta, ni siquiera a quien prometa «rescatar» la transferencia (capítulo 6).

Para pensar



1. La tienda esperaba USDT en Ethereum y los recibió en BNB Smart Chain en su misma dirección. La clave la tiene Artyom. ¿Qué hacer?
2. Un bloque de Base apareció a los 0,2 segundos. ¿Se puede considerar definitivo un pago grande?
3. El cliente envió USDC.e en lugar de USDC. ¿Es lo mismo?

Respuestas: 1) cambiar la billetera a BNB Smart Chain; para reenviarlos hace falta BNB. 2) No, espera a finalized. 3) No, es otro token: la tienda decide si lo acepta.

Lo esencial

- Las redes EVM se parecen, pero son cuadernos distintos: comprueba el chain ID.
- Hay redes independientes y segundos pisos sobre Ethereum.
- Un bloque rápido todavía no es definitivo.

A continuación: capítulo 17. TRON: energía en lugar de gas y el camino favorito de las stablecoins.

Capítulo 17. TRON: energía en lugar de gas y el camino favorito de las stablecoins

«¿Puede ser por TRC-20?»

Un cliente pregunta: «¿Aceptan USDT por TRC-20?». Artyom lo oye ya por tercera vez en la semana. ¿Qué es TRC-20 y por qué todos lo piden?

Entendamos: una red independiente

TRON es una blockchain aparte con su propia moneda, **TRX**. Su fracción más pequeña se llama **sun**: un TRX tiene un millón de sun.

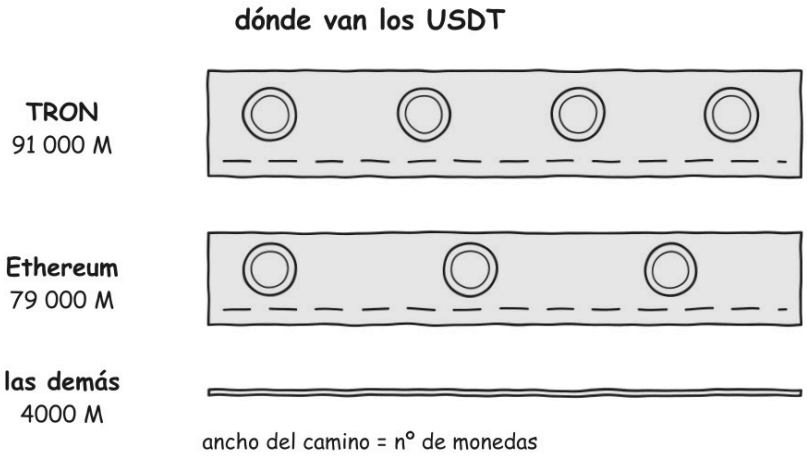
TRON está construido en muchos aspectos como Ethereum: saldos en cuentas, contratos inteligentes, eventos. Los programas los ejecuta su propia máquina virtual, pariente cercana de la EVM del capítulo 15.

TRC-20 son las reglas con las que en TRON se crean tokens, el hermano del estándar ERC-20 de Ethereum. «USDT por TRC-20» significa: el token USDT en la red TRON.

Los bloques salen cada 3 segundos. Los escriben 27 representantes elegidos (capítulo 8).

El camino favorito de las stablecoins

USDT vive en muchas redes, pero de forma muy desigual. Estas son las cifras de la propia Tether, la empresa que emite la moneda, consultadas el 27 de noviembre de 2026:



Más de la mitad de todos los USDT están en TRON. Por eso los clientes eligen tan a menudo precisamente esta red.

Fuente: app.tether.to/transparency.json. Se contaron las monedas emitidas menos la reserva todavía no puesta en circulación.

La dirección empieza por T

Por dentro, una dirección de TRON son 20 bytes, como en Ethereum, con el prefijo `41`. A las personas se la muestran de otra forma, empezando por la letra `T`:

Así la ven las personas	Así es por dentro (dirección del contrato de USDT)
`TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t`	`41a614f803b6fd780986a42c78ec9c7f77e6ded13c`

En la forma con `T` va escondida una suma de control: se puede detectar una errata. Preguntemos al nodo por una dirección auténtica y después cambiémosle la última letra:

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

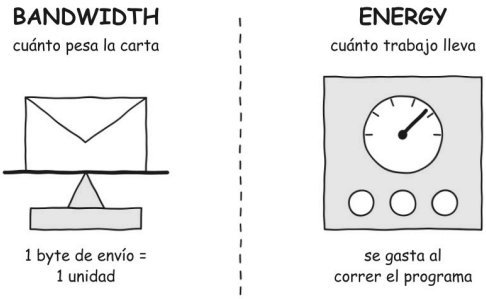
curl https://public-tron-mainnet.rpcnode.dev/wallet/validateaddress \
  -d '{"address":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6u","visible":true}'
```

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

`"visible":true` significa «escribo las direcciones en la forma con T».

Dos pagos en lugar de gas

En el capítulo 7 prometimos contar cómo calcula TRON el «peso» de una transferencia. Lo calcula con dos medidas.



- **Bandwidth**, por el espacio. El consumo es aproximadamente igual al tamaño de la transferencia en bytes. Lo paga cualquier transferencia.

- **Energy** (energía), por el trabajo del programa. Hace falta cuando se llama a un contrato, y por tanto también al transferir USDT.

Una dirección que haya recibido TRX al menos una vez tiene un regalo: **600 unidades de bandwidth al día**. Lo gastado se recupera poco a poco, en 24 horas. No hay energía gratis.

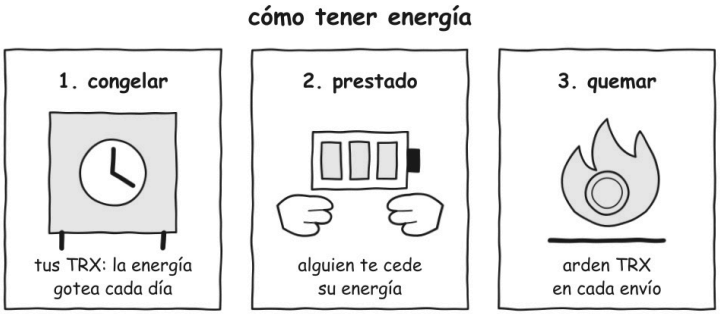
Si falta algo, la red toma TRX y los quema. ¿A qué precio? Preguntémoslo al nodo:

```
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
```

Es decir, una unidad de bandwidth cuesta 1000 sun y una de energía, 100 sun. Estos precios se cambian por votación de los representantes.

De dónde sacar energía

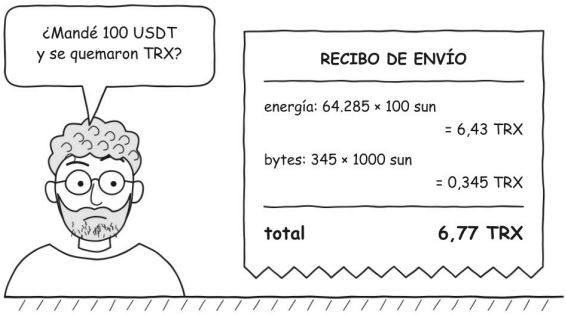


- **Congelar TRX** (en inglés, staking). Las monedas siguen siendo tuyas y la dirección recibe cada día una parte de la energía total. Para recuperarlas hay que esperar 14 días después de descongelarlas.
- **Recibirla prestada:** el dueño de TRX congelados cede su energía a otra dirección.
- **No hacer nada:** lo que falte se pagará con TRX quemados.

¿Cuánto congelar? Depende de cuántas monedas hayan congelado todos los demás. El día de la comprobación, para una transferencia de USDT al día habrían bastado unos 6600 TRX congelados.

El recibo de una transferencia real

Esta es una transferencia de 100 USDT del bloque 88 352 467. Quien envió no tenía ni energía ni bandwidth libre:



El recibo se obtiene con el método `gettransactioninfobyid`. En el nodo público está cerrado y hace falta un endpoint personal, así que esto es un esbozo:

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

Campo de la respuesta	Valor	Qué significa
`energy_usage_total`	64 285	energía gastada en total
`energy_penalty_total`	49 635	de ella, el recargo por popularidad
`energy_fee`	6 428 500	quemado por energía, en sun
`net_fee`	345 000	quemado por bytes, en sun
`fee`	6 773 500	total, es decir, 6,7735 TRX
`result`	`SUCCESS`	el programa terminó con éxito

¿Qué recargo es ese? Cuando un mismo contrato se usa demasiado, la red encarece sus llamadas. Ese día, en USDT se iba casi 4,4 veces más energía que sin el recargo.

Y además: si el destinatario no tenía USDT hasta entonces, se necesita el doble de energía, 130 285. El contrato tiene que abrirle una línea nueva.

Bloque reciente y bloque definitivo

Los nodos de TRON tienen dos juegos de direcciones. `/wallet/…` responde con los datos más recientes. `/walletsolidity/…`, solo con los definitivos. Un bloque se vuelve definitivo cuando 19 de los 27 representantes han publicado sus bloques a su altura o por encima. Normalmente es cerca de un minuto.

El día de la comprobación, el bloque reciente era el 88 352 763 y el definitivo, el 88 352 746.

Para experimentos existe la red de prueba Nile, que tiene su propio nodo público: `public-tron-nile.rpcnode.dev`.

La trampa: «si está en un bloque, ha llegado»

Miremos las transferencias directas de USDT en el último bloque 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('transferencias de USDT:', len(rows))
print('no realizadas:', sum(r[1] != 'SUCCESS' for r in rows))
for r in rows[:3]:
    print(*r)
EOF
```

A ti te saldrán otros números. En el bloque 88 352 467, de 41 transferencias así, dos no se realizaron. Una de ellas era de 100 000 USDT:

```
fbfec9fa OUT_OF_ENERGY 100000.0
```

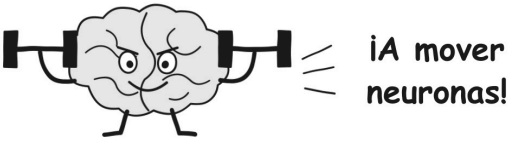
`OUT_OF_ENERGY`: no alcanzó la energía, el programa se detuvo y los USDT no se fueron a ninguna parte. Pero la anotación está en el bloque, y en una captura de pantalla la cantidad impresiona.

Cuánta energía se permite gastar lo fija quien envía: es el campo `fee_limit`, un límite en sun. Si el límite no alcanza, la transferencia no se realiza y lo quemado no se devuelve.

Qué debe recordar la tienda

- Espera el bloque definitivo y comprueba `result` en el recibo.
- Busca en el recibo el evento `Transfer` del contrato `TR7NHq…Lj6t`. Así verás también las transferencias hechas desde otros programas. USDT tiene seis decimales.
- Para reenviar USDT desde tu dirección necesitas en ella TRX o energía. Una dirección por pedido es cómoda, pero juntar el dinero de cien direcciones son cien transferencias de pago.
- La primerísima transferencia de TRX a una dirección nueva le cuesta a quien envía al menos 1 TRX más.

Para pensar



1. Una dirección tiene 600 de bandwidth gratis y mucha energía. Envía seguidas dos transferencias de 345. ¿Qué pasa con la segunda?
2. No hay energía y el destinatario nunca ha tenido USDT: se necesitan 130 285 unidades a 100 sun. ¿Cuántos TRX se queman por la energía?
3. El cliente envió el hash. La transacción está en un bloque y dice `OUT_OF_ENERGY`. ¿Se envía el café?

Respuestas: 1) quedan 255 gratis, que no alcanzan para 345; se queman 0,345 TRX por los 345 completos, no se suman por partes. 2) 13,0285 TRX. 3) No, no hubo transferencia.

Lo esencial

- TRON tiene dos medidas: bandwidth por los bytes y energy por el trabajo del programa.
- Si no hay energía, se queman TRX, y por USDT bastante más que por un simple envío.
- Un pago es una transferencia con éxito del USDT auténtico en un bloque definitivo.

A continuación: capítulo 18. Solana: todo repartido en cuentas.

Capítulo 18. Solana: todo repartido en cuentas

«¿Dónde están mis USDC?»

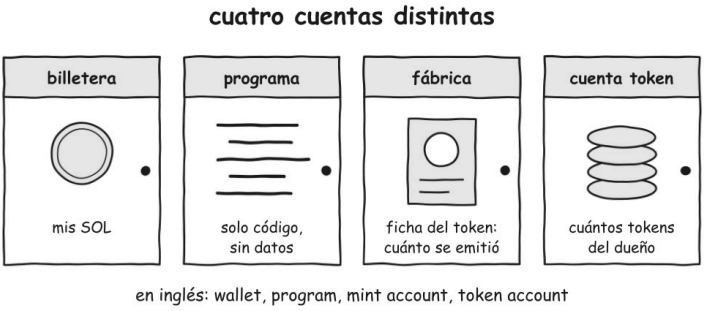
Un cliente pagó en USDC. Artyom le pide al nodo el saldo de su billetera y solo ve unas migajas de SOL. Ni rastro de USDC. ¿Lo han engañado?

Entendamos: una red hecha de cuentas

Solana es una red independiente con la moneda **SOL**. Su fracción más pequeña se llama **lamport**: un SOL tiene mil millones.

Una dirección de Solana son 32 bytes escritos con letras y cifras. No tiene un comienzo especial (capítulo 6) ni tampoco una suma de control por dentro.

Todo lo que guarda la red está en **cuentas** (accounts). Aquí una cuenta no es solo dinero, sino cualquier casilla con una dirección. En Ethereum el contrato guarda sus datos dentro de sí. En Solana **el programa va por un lado y los datos por otro**.



Cada cuenta tiene un programa dueño (el campo `owner`). Solo él puede cambiar los datos de la cuenta.

Probemos: el pasaporte de un token

Cada token tiene una cuenta-pasaporte, llamada **mint**. Pidamos el pasaporte de 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"}]}'
```

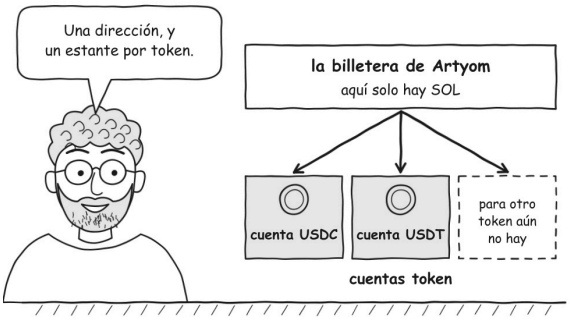
Esto es lo principal de la respuesta:

Campo	Valor	Qué significa
`owner`	`Tokenkeg…Q5DA`	la cuenta la maneja el programa de tokens
`supply`	`8513455604386747`	cuánto se ha emitido, en fracciones pequeñas
`decimals`	6	o sea, unos 8,5 mil millones de USDC
`mintAuthority`	`BJE5…5ruG`	quién puede emitir más
`freezeAuthority`	`7dGb…Crar`	quién puede congelar (capítulo 9)

USDC no tiene programa propio. Un solo programa de tokens común atiende a miles de tokens, que se distinguen por sus pasaportes. Programas comunes así hay dos: el normal, **Token** (en él están USDC y USDT), y otro más nuevo, **Token-2022**, con funciones adicionales.

La cuenta de token: un estante aparte

Cuántos USDC tiene Artyom no está anotado ni en la billetera ni en el pasaporte, sino en una **cuenta de token**. Cada token tiene la suya.



La cuenta de token principal se llama **ATA** (associated token account). Su dirección se calcula a partir de tres cosas: la dirección de la billetera, el pasaporte del token y el programa de tokens. Por eso al cliente se le da la **dirección de la billetera**, y su programa encuentra el estante por sí mismo.

Miremos una cuenta de token de verdad. Primero preguntamos cuántos tokens hay en ella y después cuántos SOL:

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

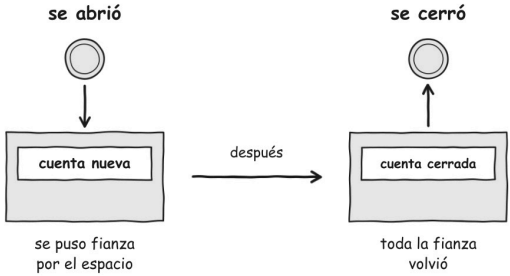
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getBalance","params":
  ["BnGihgNUu4X1JdBMNhbWnXUcpib8UQQC154isHTYwDNK"]}'
```

En la primera respuesta, `"uiAmountString":"207637.336362"` son USDT. En la segunda, `"value":2039280` son lamports, en total 0,00203928 SOL. El método `getBalance` **no ve los tokens**. Ahí es donde se «perdieron» los USDC de Artyom.

Por cierto, el dueño de este estante es la dirección del propio pasaporte de USDT. Parece que la gente envió USDT por error «a la dirección de USDT», como en el capítulo 9.

El depósito por el espacio

Cada cuenta ocupa memoria en todos los nodos. Por eso lleva un **depósito** en SOL (en inglés rent, «alquiler», aunque no es un alquiler).



Una cuenta de token ocupa 165 bytes. ¿Cuánto depósito hace falta para eso?

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

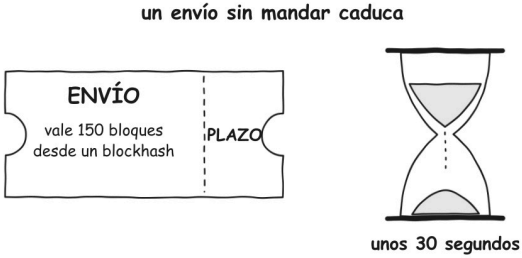
```
{"jsonrpc":"2.0","result":1488440,"id":1}
```

Son unos 0,0015 SOL. En 2026 el depósito se está rebajando por etapas, así que en el programa no se escribe el número, sino que se pregunta. Si el destinatario aún no tiene estante para el token, quien envía suele crearlo en la misma transferencia y pone él el depósito.

Comisión y fecha de caducidad

La comisión base son **5000 lamports por cada firma**: en una transferencia normal con una firma son 0,000005 SOL. La mitad se quema. Encima se puede añadir un pago por urgencia, que se lleva entero el validador. La comisión se cobra aunque la transferencia falle.

En cada transferencia se incluye un **blockhash**, la huella de un bloque reciente. Él fija además la fecha de caducidad.




```
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"}]}'
```

En la primera respuesta llegó `"lastValidBlockHeight":452330099`; en la segunda, un segundo después, la altura 452 329 960. Quedaban 139 bloques. Si el plazo venció y la transferencia no está en ningún bloque, ya no pasará. Entonces se puede firmar otra. La excepción son unas transferencias diferidas especiales (durable nonce), que no tienen ese plazo.

2026: todavía más rápido

Qué cambió	Antes	Ahora
Duración del slot, es decir, el turno para un bloque (octubre)	0,4 s	0,2 s
Finalidad, nuevo sistema de votación Alpenglow (octubre)	unos 13 s	unos 0,15 s
Tamaño máximo de una transacción, nuevo formato v1 (septiembre)	1232 bytes	4096 bytes

Comprobemos la velocidad por nuestra cuenta. El método devuelve cuántos slots pasaron en un minuto:

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

A nosotros nos llegó `numSlots` = 277 en 60 segundos: aproximadamente 0,22 s por slot. Hay slots vacíos, así que el número de slot y la altura de bloque son cifras distintas; no las confundas.

El nodo puede responder con distinto grado de exigencia: `processed`, el bloque se vio, pero todavía se puede saltar; `finalized`, definitivo. Hay además una palabra intermedia, `confirmed`: tras Alpenglow significa lo mismo que `finalized`. El pedido se envía con `finalized`.

La trampa: «la transacción no se puede leer»

Por el nuevo formato v1, a los métodos `getTransaction` y `getBlock` hay que añadirles `maxSupportedTransactionVersion` con el valor 1. Sin eso, las transacciones nuevas darán un error. El método `getTransaction` está cerrado en el nodo público, así que esto es un esbozo para un endpoint personal:

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

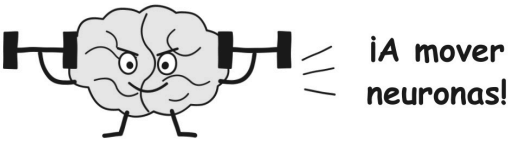
En Solana la «firma de la transferencia» sirve como su número, en lugar del hash de la transacción.

Qué comprueba la tienda

- La transacción tuvo éxito: `meta.err` es `null`.
- La cuenta de token de **tu billetera** con el pasaporte de token correcto creció en la cantidad correcta: compara `preTokenBalances` y `postTokenBalances`.
- Es el pago de ese pedido precisamente. Para eso se añade a la transferencia una **reference**, una dirección de un solo uso que marca el pedido.
- Esa firma no se ha contabilizado todavía.

Y cuidado con los SOL corrientes: una transferencia a una dirección equivocada se ejecuta igual. Lo más probable es que no haya quien devuelva ese dinero.

Para pensar



1. La billetera firmó una transferencia y la red la vio dos minutos después. ¿Qué pasará?
2. La tienda consulta `getBalance` de su billetera para detectar un pago en USDC. ¿Cuál es el error?
3. Artyom envía USDC a un proveedor que aún no tiene ese estante. Además de la comisión, ¿qué tendrá que poner?

Respuestas: 1) el plazo venció; la transferencia no pasará. 2) `getBalance` *solo muestra SOL; los USDC están en la cuenta de token. 3) El depósito de la nueva cuenta de token, unos 0,0015 SOL.*

Lo esencial

- En Solana todo son cuentas; los programas y sus datos van separados.
- Los tokens están en cuentas de token: una por cada token, con depósito.
- Lee las transacciones con `maxSupportedTransactionVersion` y espera `finalized`.

A continuación: capítulo 19. TON: cartas en lugar de llamadas.

Capítulo 19. TON: cartas en lugar de llamadas

«Lo envié, ¿y ahora qué?»

Un cliente envió USDT en la red TON. En el explorador, en lugar de una sola anotación, Artyom ve toda una cadena: cinco transacciones, unas direcciones ajenas y un cambio menudo. ¿Es un pago o son cinco?

Entendamos: todos son contratos y todos escriben cartas

TON (The Open Network) es una red independiente. Su moneda se llama **Gram** (GRAM). Antes se llamaba Toncoin, y ese nombre todavía aparece a menudo. En un Gram hay mil millones de **nanograms**.

Desde abril de 2026 los bloques salen aproximadamente cada 0,4 segundos, y una anotación se vuelve definitiva en cerca de un segundo.

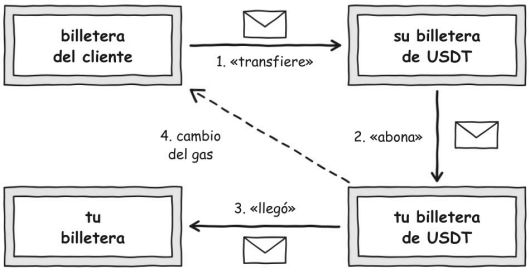
La principal diferencia es esta. Aquí cada cuenta es un contrato inteligente, y **la billetera también**. Los contratos no se llaman directamente unos a otros, sino que se envían **mensajes**.

En Ethereum, todo lo que pone en marcha una transferencia ocurre dentro de una sola transacción: o entero o nada. Es como una conversación telefónica. En TON es correspondencia: envías la carta y no te quedas esperando en la línea. El destinatario atiende cada carta más tarde, en una transacción aparte. Ese sistema se llama **asíncrono**.

Toda la correspondencia que nace de un solo encargo se llama **trace** (rastro). El éxito de una de sus transacciones no significa todavía que todo el asunto haya salido bien.

Cómo viajan los USDT

En TON los tokens se llaman **jetton**. Cada uno tiene un contrato principal (**master**), y cada propietario, su propia **billetera jetton** aparte. Se parece a las cuentas de token del capítulo 18, solo que es un contrato independiente. Para USDT el contrato principal es `EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs`.



cada carta se procesa en una transacción aparte

No es un invento, sino una transferencia real de 11,99 USDT hecha en la noche del 27 de noviembre de 2026:

De quién es la transacción	Qué recibió	Qué envió después
billetera del remitente	el encargo firmado	«transfiere» y 0,05 Gram para gastos
su billetera USDT	«transfiere»	«abona»
billetera USDT del destinatario	«abona»	el aviso y el cambio
billetera del destinatario	el aviso y 1 nanogram	nada
billetera del remitente	el cambio, unos 0,0494 Gram	nada

Todo cupo en un segundo. El dinero para gastos se adjunta con margen y el resto vuelve.

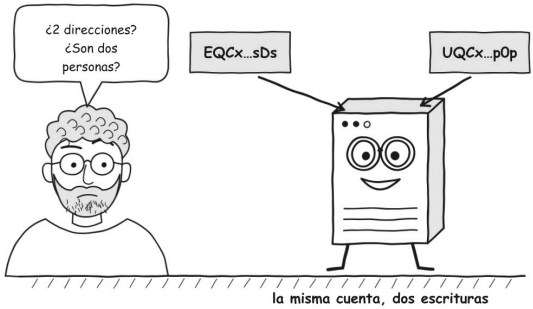
Un detalle curioso: la cuarta transacción está marcada como interrumpida (`aborted`). Al destinatario no le alcanzó el gas para ejecutar el programa: solo había llegado un nanogram. Y sin embargo los USDT ya se habían abonado en la tercera transacción. No se puede juzgar por una sola anotación, ni en un sentido ni en el otro.

La cadena entera se puede ver con una consulta al indexador. En el nodo público está cerrado, así que esto es un esbozo para un endpoint personal:

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

Una cuenta, dos direcciones

El nombre auténtico de una cuenta es la forma «cruda»: el número de la parte de la red y 64 caracteres. A las personas se les muestra otra escritura de 48 caracteres con una suma de control. Y una misma cuenta tiene varias escrituras así.



Preguntemos al nodo por la dirección de USDT:

```
curl -s "https://public-ton-mainnet.rpcnode.dev/api/v2/detectAddress?address=EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixR
v7Nw2Id_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…` : dirección con rebote (bounceable); así se escriben los contratos normales.
- `UQ…` : dirección sin rebote; así se aconseja escribir las billeteras.
- En la red de prueba las escrituras empiezan por `kQ` y `0Q` .

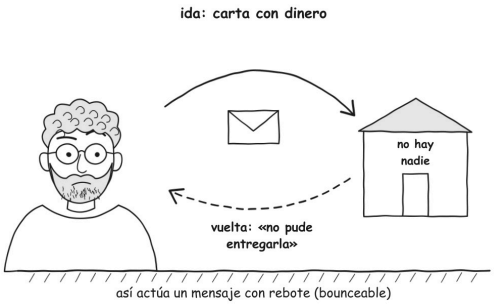
Por eso en la base de datos de la tienda **se compara la forma cruda**, no las cadenas con `EQ` y `UQ` .

La suma de control atrapa las erratas. Si al final escribes `sDt` en lugar de `sDs` , el nodo contestará con el código 422:

```
curl -s -o /dev/null -w '%{http_code}\n' \
  "https://public-ton-mainnet.rpcnode.dev/api/v2/detectAddress?address=EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2I
d_sDt"
```

Una carta a una casa vacía

¿Y si el destinatario no pudo atender la carta o su contrato todavía no existe? Depende de la marca que lleve el mensaje.



Un mensaje con rebote vuelve al remitente con el resto del dinero, si alcanza para el viaje de vuelta. Un mensaje sin rebote se queda en manos del destinatario. Una billetera nueva puede no estar todavía «desplegada», es decir, su programa no está escrito en la red. Por eso a las billeteras se les paga a la dirección sin rebote.

Probemos: comprobar la velocidad

Averigüemos el número del bloque principal más reciente y la hora a la que salieron él y el que está mil bloques antes:

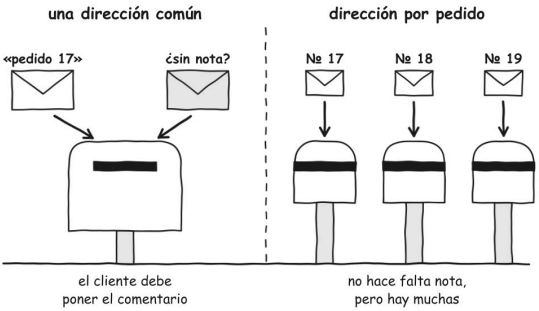
```
B=https://public-ton-mainnet.rpcnode.dev
last=$(curl -s $B/api/v2/getMasterchainInfo |
  python3 -c "import json,sys; print(json.load(sys.stdin)['result']['last']['seqno'])")
for s in $last $((last-1000)); do
  sleep 2
  curl -s "$B/api/v2/getBlockHeader?workchain=-1&shard=-9223372036854775808&seqno=$s"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

```
108011078 1795732480
108010078 1795732076
```

Los segundos números son la hora en segundos. La diferencia es de 404 segundos para 1000 bloques, es decir, unos 0,4 segundos por bloque.

Un comentario en lugar de una dirección aparte

A una transferencia en TON se le puede adjuntar un **comentario** de texto. De ahí salen dos formas de saber de quién es un pago:



Con una dirección común hay menos trabajo, pero el cliente puede olvidar el comentario o copiarlo con errores. Además, no todas las billeteras permiten añadirlo. Esos pagos se resuelven a mano.

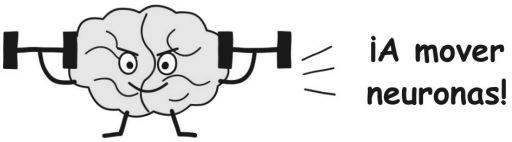
La trampa: «llegó el aviso, así que envío»

Cualquiera puede escribir un contrato que mande a tu billetera un aviso igual. O emitir su propio jetton con el nombre «USDT». Por eso se comprueba que:

- el aviso llegó de **tu** billetera USDT. Su dirección se le pregunta de antemano al contrato principal auténtico (`get_wallet_address`);
- la cantidad coincide (USDT tiene seis decimales);
- el comentario o la dirección corresponden a este pedido, y este pago no se ha contabilizado todavía;
- la anotación entró en la cadena principal (masterchain), es decir, es definitiva.

El aviso solo llega si el remitente adjuntó para ello al menos un poco de Gram. Es lo habitual, pero un programa casero puede olvidarlo. Entonces los tokens llegan sin aviso.

Para pensar



1. En la base de datos hay una dirección que empieza por `UQ`, y el explorador muestra una que empieza por `EQ` con la misma parte central. ¿Cómo comprobar si es la misma cuenta?
2. La primera transacción de la cadena tuvo éxito. ¿Significa que los USDT llegaron?
3. El cliente pagó a la dirección común, pero olvidó el comentario. ¿Qué debe hacer la tienda?

Respuestas: 1) pasar ambas a la forma cruda (`detectAddress`) y compararlas. 2) Todavía no: hace falta que la billetera USDT del destinatario los abone. 3) No darlo por válido automáticamente, sino revisarlo a mano: la cantidad, el remitente y el número de la transferencia.

Lo esencial

- En TON los contratos se escriben cartas: un encargo engendra una cadena de transacciones.
- Una misma cuenta tiene varias escrituras de la dirección; compara la forma cruda.
- Un jetton se comprueba por el contrato principal y por tu billetera jetton.

A continuación: capítulo 20. Otros enfoques: Sui y Aptos, XRP Ledger y Stellar, Cardano, Polkadot y Cosmos.

Capítulo 20. Otros enfoques: Sui y Aptos, XRP Ledger y Stellar, Cardano, Polkadot, Cosmos

«¿Y aceptan XLM?»

Un cliente pregunta si puede pagar por Stellar. Otro quiere enviar USDC en Sui. Artyom ya se las ha visto con Bitcoin y Ethereum. ¿Le bastará con eso? No del todo: aquí rigen otras costumbres.

Entendamos: cinco preguntas para una red nueva

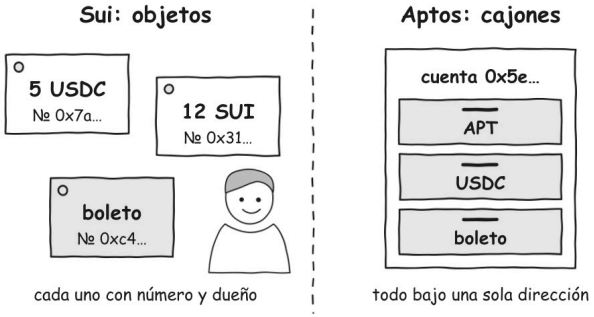
No hace falta aprenderse entera una red desconocida. Para empezar bastan cinco respuestas:

1. ¿Dónde está el dinero: en una cuenta o en trozos separados?
2. ¿Qué hay que hacer de antemano para poder siquiera recibir una transferencia?
3. ¿Cómo saber por qué pedido ha llegado?
4. ¿Cuándo es definitiva?
5. ¿Con qué se paga la comisión cuando haya que reenviar el dinero?

Sui y Aptos: un lenguaje común, distinta construcción

En las dos redes los programas se escriben en el lenguaje **Move**. Este vigila que algo valioso no se pueda copiar sin más. Pero el dinero se guarda de distinta manera.

- **Sui** lo guarda todo en **objetos**: son cosas, cada una con su número. El dinero puede estar en monedas-objeto separadas, casi como los trozos de Bitcoin. Desde 2026 una dirección puede tener además un saldo común. El saldo completo es la suma de lo uno y lo otro.
- **Aptos** se parece más a la cuenta habitual. Bajo la dirección se apilan las pertenencias, llamadas **recursos**. No hace falta abrir la cuenta de antemano.



En ambas redes la transferencia suele ser definitiva en menos de un segundo. La comisión se paga con su propia moneda: SUI o APT. En Sui hay una excepción: desde 2026, algunas stablecoins, entre ellas USDC, se pueden enviar sin comisión (desde 0,01).

La dirección en las dos es `0x` y normalmente 64 caracteres. El comienzo es como en Ethereum, pero la longitud es otra: allí, después de `0x` hay 40 caracteres.

Probemos: encontrar USDC en dos redes

RpcNode no tiene nodos públicos gratuitos para las redes de este capítulo. Así que preguntaremos en las direcciones abiertas de quienes desarrollan esas redes. Son compartidas y tienen restricciones: sirven para conocerse, no para una tienda.

Sui, con el método llamado GraphQL (enumeras qué campos necesitas):

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

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

Aptos, el REST normal del capítulo 11:

```
curl -s https://api.mainnet.aptoslabs.com/v1/accounts/0xbae207659db88bea0cbead6da0ed00aac12edcdda169e591cd41c94180
b46f3b/resource/0x1::fungible_asset::Metadata
```

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

En un sitio y en otro, el token no se encuentra por la palabra «USDC», sino por una larga dirección de la lista del emisor, la empresa Circle. Es la regla del capítulo 9. En ambos casos hay seis decimales.

Cuidado con los ejemplos antiguos. Antes a Sui se le preguntaba por JSON-RPC. En julio de 2026 se desactivó en los nodos de la fundación Sui; ahora allí hay gRPC y GraphQL. Antes de copiar un ejemplo de un artículo, mira su fecha.

XRP Ledger y Stellar: redes de transferencias

Las dos redes saben desde hace tiempo no solo enviar su propia moneda, sino también llevar tokens ajenos. La página de la cadena se llama aquí **ledger**. XRP Ledger aprueba una nueva cada 3–5 segundos; Stellar, aproximadamente cada 5 segundos. Lo aprobado no se reescribe.

Pero hay dos reglas poco habituales.

La primera: un depósito por el espacio. La cuenta no existe mientras no tenga una cantidad mínima, la **reserva**. Y para recibir un token, por ejemplo USDC, el dueño de la cuenta abre de antemano una **línea de confianza** (trust line): «sí, acepto tener esto». Cada anotación así aumenta la reserva.

	XRP Ledger	Stellar
La dirección empieza por	`r`	`G`
Mínimo en la cuenta	1 XRP	1 XLM
Aumento por trust line	0,2 XRP (las dos primeras pueden librarse)	0,5 XLM
Comisión mínima	0,00001 XRP	0,00001 XLM por operación
Número de pedido	destination tag	memo o dirección que empieza por `M`

La red puede cambiar estas cifras por votación, así que se le preguntan al nodo:

```
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` es 0,00001, `1E0` es 1 y `2E-1` es 0,2. Ahora Stellar:

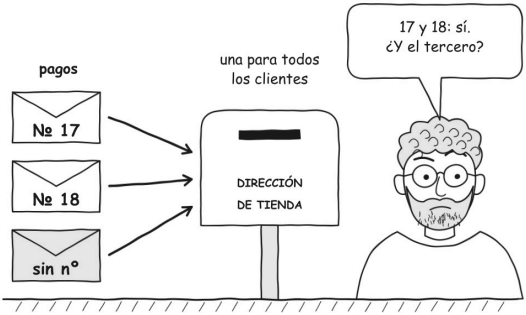
```
curl -s 'https://horizon.stellar.org/ledgers?order=desc&limit=1' \
  | grep -E '"(base_fee|base_reserve)_in_stroops"'
```

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

En un XLM hay diez millones de **stroops**. Así que la comisión es de 0,00001 XLM y una parte de la reserva, de 0,5 XLM (una cuenta necesita al menos dos).

Una dirección para todos

La segunda regla. Abrir aquí una dirección para cada pedido sale caro: cada una tendría su reserva. Por eso la tienda muestra una sola dirección y pide añadir un número: el **destination tag** en XRP Ledger, el **memo** en Stellar.

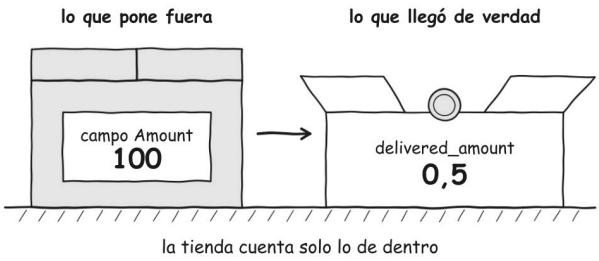


Si el cliente olvida el número, el dinero llega, pero no está claro por qué. Qué ayuda:

- en XRP Ledger se puede activar el ajuste `RequireDest`: la red no dejará pasar una transferencia sin número;
- el número se puede esconder en la propia dirección: X-address en XRP Ledger, dirección con `M` en Stellar, si la billetera del cliente las entiende.

La trampa: 100 en la etiqueta, 0,5 dentro

En XRP Ledger, quien envía puede permitir un **pago parcial** (partial payment): la transferencia se considera exitosa aunque llegue menos. Entonces el campo `Amount` no dice «llegó», sino «como máximo». Cuánto llegó de verdad está anotado en `delivered_amount`.



Esta es una anotación real de la red; preguntemos por ella con el 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"}
```

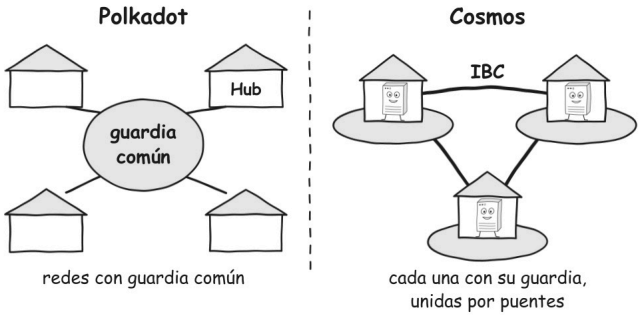

El resultado es `tesSUCCESS`, en `Amount` hay unos 9,55 y llegaron unos 4,53. Por sí sola, una anotación así no es un engaño. Pero un estafador puede escribir una suma grande, enviar una migaja y esperar la mercancía. **Cuenta solo** `delivered_amount` y comprueba qué token es y de qué emisor.

Cardano: trozos con adjuntos

Cardano está construido de forma más parecida a Bitcoin (capítulo 14): no hay saldo, sino trozos sin gastar. Solo que el trozo sabe hacer más, y el modelo se llama **UTXO extendido** (eUTXO).

- Junto a la moneda ADA, en el trozo van otros tokens. No hace falta un contrato aparte para ellos.
- Un token no viaja solo: en el mismo trozo tiene que haber un poco de ADA. Por eso con el token llegan también unas pocas monedas.
- La comisión depende del tamaño de la transferencia en bytes.
- La certeza crece poco a poco: cuantos más bloques encima, más seguro.

Polkadot y Cosmos: no una red, sino muchas



Polkadot es una cadena central que protege a las redes conectadas a ella. Un cambio importante: desde noviembre de 2025 los saldos de los usuarios ya no viven en la cadena central, sino en la red **Polkadot Hub** (también llamada Asset Hub).

- Allí están también USDT (activo número 1984) y USDC (número 1337).
- La cuenta tiene un umbral de existencia, el **existential deposit**: 0,01 DOT. Si no alcanza ni el DOT ni el USDT o USDC en la cantidad necesaria, la cuenta se borra.
- La dirección empieza por `1`, como las antiguas de Bitcoin. No las confundas.

Cosmos son redes independientes hechas con piezas comunes. Cada una tiene su propia protección, su moneda y su prefijo de dirección (en Cosmos Hub, `cosmos1…`). Los bloques son definitivos enseguida. La transferencia tiene un campo **memo** para el número de pedido.

Las redes se conectan con el protocolo **IBC**. Un token llegado de una red vecina no se llama con la palabra habitual, sino `ibc/` y un sello largo, por ejemplo `ibc/7F1D3FCF…`. El sello indica por qué camino llegó.

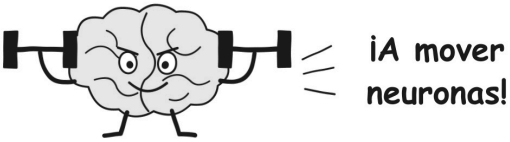
Y una stablecoin también puede marcharse. Circle anunció que deja de emitir USDC en la red Noble, de la familia Cosmos: la emisión nueva está desactivada desde el 13 de octubre de 2026, y el 12 de enero de 2027 allí detendrán USDC. A los poseedores les aconsejan transferirlo con antelación.

Todo junto

Red	Dirección	Se necesita de antemano	Definitivo
Sui	`0x…`	nada	menos de un segundo
Aptos	`0x…`	nada	menos de un segundo
XRP Ledger	`r…`	reserva, trust line	3–5 segundos
Stellar	`G…`	reserva, trust line	unos 5 segundos
Cardano	`addr1…`	nada	poco a poco
Polkadot Hub	`1…`	saldo por encima del umbral	por la marca
Cosmos Hub	`cosmos1…`	nada	rápido

La trust line solo hace falta para los tokens. Comprobado el 27 de noviembre de 2026; las reglas cambian.

Para pensar



1. Artyom creó una dirección nueva y vacía en Stellar y enseguida emitió una factura en USDC. ¿Qué está mal y cuántos XLM hacen falta como mínimo?
2. Llegó una transferencia: `Amount` es 100 USDC, `delivered_amount` es 0,5 y el resultado, `tesSUCCESS`. ¿Se envía café por valor de 100?
3. Los clientes de XRP Ledger olvidan el destination tag. ¿Qué hay que activar?

Respuestas: 1) la cuenta todavía no existe y la trust line no está abierta; hace falta una reserva de al menos 1,5 XLM y un poco para la comisión. 2) No: llegó 0,5. 3) El ajuste RequireDest.

Lo esencial

- Hazle cinco preguntas a una red desconocida antes de mostrar una dirección al cliente.
- En XRP Ledger y Stellar hacen falta reserva, trust line y número de pedido; en XRP Ledger cuenta solo `delivered_amount`.
- Las costumbres de Ethereum y los ejemplos antiguos no se trasladan a otra red.

A continuación: capítulo 21. Qué red elegir.

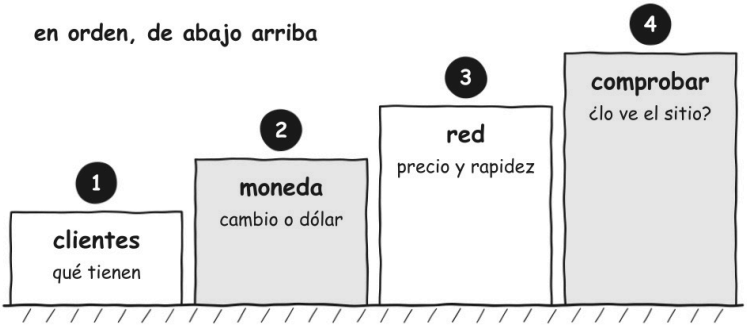
Capítulo 21. Qué red elegir

«Entonces, de todas, ¿cuál?»

En siete capítulos se han juntado muchos nombres: Bitcoin, Ethereum, TRON, Solana, TON y una decena más. Artyom tiene una sola tienda en línea y un solo taller de tueste de café. ¿Conectarlo todo? ¿Elegir una? ¿Con qué criterio?

Entendamos: cuatro pasos

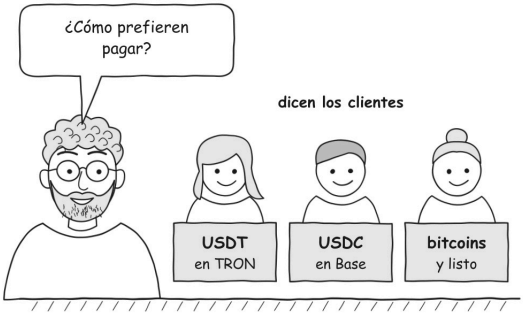
No existe «la mejor red». Existe la red que les conviene a tus clientes y a tu pedido. Conviene ir en orden.



Aquí se trata de con qué conviene cobrar. No es un consejo sobre en qué invertir dinero.

Paso 1. Pregunta a los clientes

Paga quien ya tiene las monedas. Si las tiene en una red y tú cobras en otra, lo más probable es que no haya pago.



Pregunta en el boletín, en el chat, al hacer el pedido. Bastan dos preguntas: «¿Qué tienes?» y «¿En qué red?».

Paso 2. Qué moneda

	Moneda propia de la red	Stablecoin (USDT, USDC)
Precio en dólares	cambia, a veces bruscamente	intenta mantenerse cerca de un dólar
Quién manda	las reglas de la red	hay un emisor que puede congelar (capítulo 9)
Con qué se paga la comisión	con ella misma	normalmente con la moneda de la red, no con la stablecoin

Si los precios del café de Artyom están en dólares o ligados a ellos, con una stablecoin hay menos sorpresas. Pero también es confianza: en la empresa que la emite.

Paso 3. Qué red

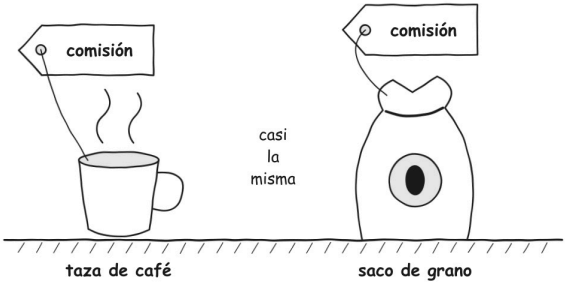
Una misma stablecoin no vive en todas partes. Estas son las redes de los capítulos 14–20. «Sí» significa que el sitio del emisor indica para esa red la dirección de precisamente esa moneda.

Red	Comisión	USDT de Tether	USDC de Circle	No olvidar
Bitcoin	BTC	no	no	confirmaciones
Litecoin, Doge y otras	propia	no	no	redes distintas
Ethereum	ETH	sí	sí	el gas
Base, Arbitrum	ETH	no	sí	finalized
BNB Smart Chain	BNB	no	no	el emisor
TRON	TRX	sí	no	la energía
Solana	SOL	sí	sí	cuentas de token
TON	Gram	sí	no	los mensajes
Aptos	APT	sí	sí	dirección larga
Sui	SUI	no	sí	gRPC, GraphQL
Polkadot Hub	DOT	sí	sí	umbral de cuenta
XRP Ledger	XRP	no	sí	reserva, etiqueta
Stellar	XLM	no	sí	reserva, etiqueta
Cardano	ADA	no	no	ADA con el token

Según tether.to/en/supported-protocols y developers.circle.com a 27 de noviembre de 2026. «No» no significa que allí no exista un token con ese nombre: comprueba quién lo emite (capítulo 9). En Cosmos todo depende de la red concreta.

El precio de la transferencia y el tamaño del pedido

La comisión no depende de la cantidad, sino del «peso» de la transferencia y de lo ocupada que esté la red (capítulo 7).



Para un saco de grano es una minucia. Para una taza de café, la comisión puede salir más cara que la propia taza. Por eso, para pedidos pequeños, se mira dónde es más barato transferir. Los precios cambian, así que se consultan en la billetera antes de enviar, no en una tabla vieja.

Y recuerda el camino de vuelta: para reenviar lo recibido, la dirección necesita moneda para la comisión. Cien direcciones son cien transferencias de pago.

Cuánto esperar

Definitivo en	Redes
segundos o menos	BNB Smart Chain, Solana, TON, Aptos, Sui, XRP Ledger, Stellar
cerca de un minuto	TRON
decenas de minutos	Ethereum, Base, Arbitrum One
cerca de una hora	Bitcoin (6 confirmaciones)

Aproximado, según los capítulos 8 y 17–20.

Probemos: qué antigüedad tiene el bloque definitivo

Pidamos a las redes EVM el bloque `finalized` (capítulo 8) y calculemos hace cuántos segundos salió:

```
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
```

En Ethereum son unos 15 minutos; en Base, unos 22. En BNB Smart Chain, un segundo. Tus cifras serán otras: la marca avanza a saltos. Para un pedido pequeño no es obligatorio esperar la marca más segura, pero entonces el riesgo es tuyo.

Paso 4. ¿Lo verá tu sitio?

Antes de publicar la dirección en el sitio, comprueba:

- si el proveedor tiene un endpoint para esa red (capítulo 13);
- si para ella en concreto hay avisos de pagos. No es lo mismo. En RpcNode se puede obtener un endpoint para más redes que aquellas que se analizan para eventos (capítulos 26 y 27);
- si hay red de prueba para ensayar sin dinero de verdad;
- si tu billetera y el sitio donde cambias las monedas manejan esa moneda **precisamente en esa red**.

Y mira el plan: en Free, RpcNode solo tiene endpoints para Ethereum y Solana (capítulo 22).

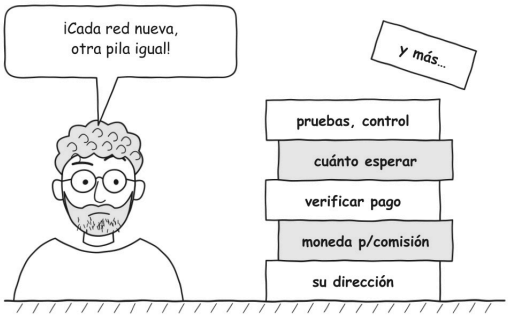
Ejemplo: cómo decide Artyom

Supongamos que preguntó a 20 clientes. Las cifras son de ejemplo; tu tienda tendrá las suyas.

Con qué quieren pagar	Cuántas personas	Decisión
USDT en TRON	9	conectar ya
USDC en Base	5	conectar ya
BTC	3	más adelante, si lo piden
ETH	2	más adelante
XLM	1	por ahora no

Dos redes cubren a 14 de 20 personas. A los demás, Artyom les escribe honradamente que todavía no acepta de todo.

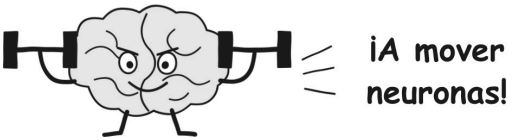
La trampa: «lo conecto todo de una vez»



Una línea más en la página de pago parece gratis. Pero detrás hay su propia dirección, su moneda para la comisión, sus reglas de comprobación y sus sorpresas tras las actualizaciones. Empieza con una o dos y añade según te pidan.

Y la segunda trampa: escribir en la página de pago simplemente «USDT». El cliente elegirá la red por su cuenta y puede equivocarse (capítulo 16). Escribe la moneda y la red juntas, en grande: «USDT en la red TRON».

Para pensar



1. Un pedido de 5 dólares, y hoy la comisión en la red elegida es de 2 dólares. ¿A quién le incomoda y qué se puede hacer?
2. Un cliente pide que aceptes USDT en Base. ¿Qué muestra la lista del sitio de Tether?
3. Artyom quiere aceptar ocho redes desde el primer día. ¿Qué le objetarías?

Respuestas: 1) al cliente: la comisión es un 40% añadido al pedido; proponer una red donde ahora transferir sea más barato. 2) Base no figura: antes de aceptar un token con ese nombre, averigua quién lo emite. 3) Son ocho pilas de preocupaciones; empezar con una o dos, las que más necesiten los clientes.

Lo esencial

- La elección va en orden: clientes, moneda, red y comprobación.
- Compara la comisión con el tamaño del pedido y ten en cuenta cuánto hay que esperar.
- Empieza con una o dos redes y nombra siempre la moneda junto con la red.

Fin de la cuarta parte. Artyom ya sabe en qué se diferencian las redes y ha decidido por dónde empezar. **A continuación:** parte 5, capítulo 22. Los primeros cinco minutos en RpcNode.

Capítulo 22. Los primeros cinco minutos en RpcNode

«Quiero mi propia puerta»

Durante todo el libro, Artyom ha llamado a los nodos públicos. Contestaban sin registro, pero esa puerta es de todos. Para la tienda hace falta una propia. ¿Se tarda mucho en hacerla? ¿Qué habrá que rellenar?

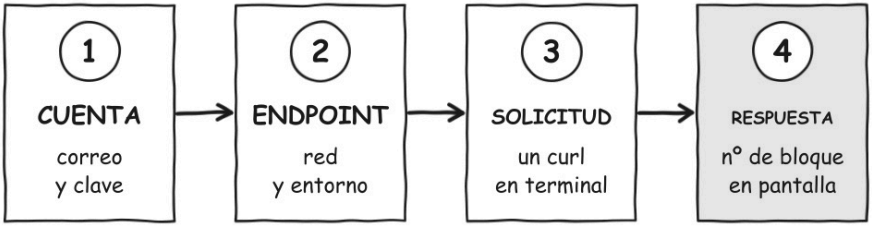
Entendamos: la puerta común y la propia

La dirección a la que un programa hace preguntas al nodo se llama **endpoint** (capítulo 3). En RpcNode los hay de dos tipos.



	Dirección pública	Endpoint personal
Registro	no hace falta	hace falta
Clave	no	incrustada en la URL
Límite	5 solicitudes por segundo por IP	según el plan
Métodos pesados	cerrados	abiertos
Archivo (capítulo 24)	cerrado	en los planes de pago
WebSocket y gRPC	no	donde la red lo admite

Tu propio endpoint aparece en cuatro pasos.



Paso 1. Registro

Abre `cabinet.rpcnode.dev/register` o pulsa «Empezar» en el sitio rpcnode.dev. El panel es tu página dentro del servicio: allí están los endpoints, el consumo y el plan.

1. Escribe el **Correo electrónico**, inventa una **Contraseña** y repítela. El nombre se puede dejar vacío.
2. Lee y marca «Acepto los términos del servicio». Supera la comprobación de «no soy un robot» si aparece.
3. Pulsa «Continuar».
4. Al correo llegará un código de un solo uso. Introdúcelo y pulsa «Verificar y continuar».

También hay un camino corto: entrar con Google o GitHub.

La cuenta nueva recibe el plan gratuito **Free**, sin tarjeta. Incluye 15 millones de créditos al mes, 15 solicitudes por segundo, 3 endpoints y dos redes: Ethereum y Solana. Qué son los créditos lo veremos en el capítulo 25.

Paso 2. El endpoint

En el menú del panel elige «Endpoints» y pulsa «Nuevo endpoint». Se abre una ventana de tres pasos:

Paso en la ventana	Qué hacer
Elige la red	por ejemplo, Ethereum
Elige el entorno	red principal o de prueba (más abajo hablamos de ellas)
Confirmar	revisar y pulsar «Crear»

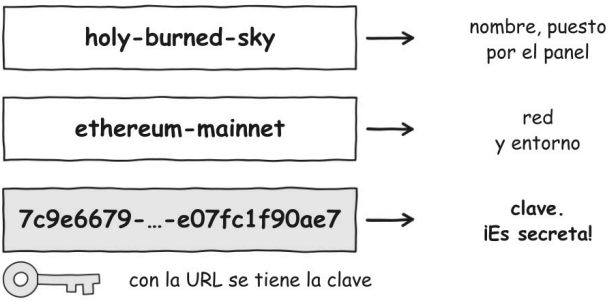
El **entorno** es la variante de la red: la real o la de prácticas. El nombre del endpoint lo inventa el propio panel, con tres palabras inglesas.

En el plan Free, antes de crearlo verás una advertencia sobre los nodos de archivo. No es un error; de eso trata el capítulo 24.

De qué se compone la URL

Este es un ejemplo de la documentación de RpcNode. Es una sola línea; aquí está partida antes de la barra:

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



La larga cola que va detrás de la barra se llama **UUID**: es un número prácticamente imposible de adivinar. Esa es la clave. No hace falta adjuntar a la solicitud ninguna contraseña aparte.

Copia la URL con el botón del panel, no la teclees. En pantalla se oculta parte de la clave, pero se copia entera.

En el menú está además «Claves de API». Es otra cosa: con ellas se firman las órdenes al propio servicio, por ejemplo «crea un pago». Para las preguntas normales al nodo no hacen falta.

Paso 3. Probemos: la primera solicitud

Ponemos la URL en un ajuste y preguntamos el número del último bloque. Esto es un **esbozo**: en lugar del texto, pon tu dirección.

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

La URL personal solo la tienes tú, así que en el libro mostraremos la respuesta a esas mismas líneas con la pública: `RPC_URL='https://public-ethereum-mainnet.rpcnode.dev'`.

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

`0x19464e4` es el bloque número 26 502 372. En el capítulo 3 el nodo decía 26 502 035: a la red le ha dado tiempo de sumar 337 bloques.

Para Solana escribe `getSlot` en lugar de `eth_blockNumber`.

En la página «Resumen» del panel hay una lista «Primeros pasos». En ella está el comando preparado, ya con tu URL, y el letrero «Esperando la primera solicitud…». Y en la página del endpoint, en la pestaña «En vivo», se ven las propias solicitudes: hora, método, respuesta y latencia.

Si no salió

Preguntemos a propósito con una clave inventada, toda de ceros:

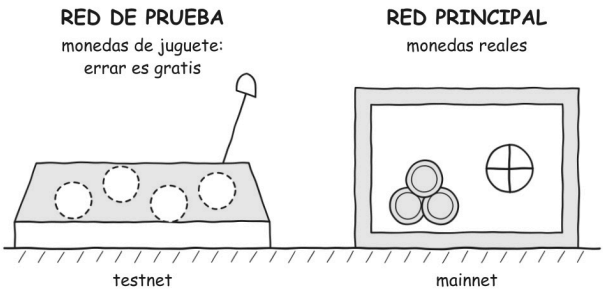
```
HOST=https://holy-burned-sky-ethereum-mainnet.rpcnode.dev
KEY=00000000-0000-0000-0000-000000000000
curl -s -w '  HTTP %{http_code}\n' $HOST/$KEY \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
```

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

`endpoint not found`: «no existe ese endpoint». Comprueba que copiaste la URL entera. No repitas una URL errónea en bucle: si insistes probando, dejarán de escucharte durante un tiempo.

Redes de prueba: un campo de entrenamiento con dinero de juguete

Una **red de prueba** (testnet) funciona con las mismas reglas que la real, pero sus monedas no valen nada. A la real se la llama **mainnet**. Aquí te puedes equivocar gratis.



Las monedas de prueba se sacan de un «grifo» (faucet): una página que las envía a la dirección que le indiques.

Red	Entornos de prueba en RpcNode, por ejemplo
Ethereum	Sepolia, Hoodi
Solana	testnet, devnet
Bitcoin	testnet4, signet
TRON	Nile, Shasta

Para las aplicaciones sobre Ethereum, el sitio ethereum.org aconseja Sepolia. Hoodi es para quienes prueban la propia red y los validadores.

Hay también redes de prueba sin registro: a 27 de noviembre de 2026, en la lista de nodos públicos están Bitcoin testnet4, TRON Nile y TON testnet. Preguntemos a TRON, la real y la de prueba, su número de red (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"}
```

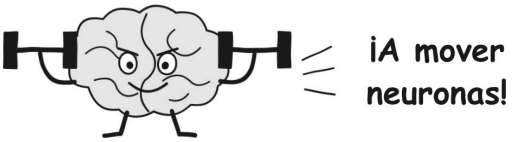
Los números son distintos: son dos cuadernos diferentes. El endpoint también está atado a una red y un entorno. Para pasar de la de prueba a la real se crea uno nuevo.

La trampa: «pego la URL directamente en la página del sitio»

Todo lo que llega al código de la página lo ve cualquier visitante. Tomará tu URL y gastará tu límite.

- Pregunta al nodo desde tu **servidor**, no desde el navegador del cliente.
- Guarda la URL en la configuración del servidor, no en el código ni en un repositorio compartido.
- En la página del endpoint, en «Ajustes», define la «Lista de IP permitidas»: así una computadora ajena recibirá un rechazo.
- Los «Orígenes permitidos» impiden llamar al endpoint desde un navegador en sitios ajenos. Pero esa lista no protege contra un programa que funcione fuera del navegador.
- ¿Se filtró la URL? Crea otra, cambia el sitio a ella y borra la antigua: no se podrá recuperar.

Para pensar



1. Artyom escribió la URL del endpoint en un script de la página de la tienda para mostrar el número de bloque. ¿Quién verá la clave?
2. Todo está probado en Sepolia. ¿Se pueden cobrar pagos reales con ese mismo endpoint?
3. Llega `endpoint not found`. ¿Por dónde empezar a buscar?

Respuestas: 1) cualquier visitante. 2) No: hace falta un endpoint en mainnet. 3) Cotejar la URL con la del panel: volver a copiarla entera.

Lo esencial

- El endpoint personal es una URL con la clave dentro: cuídala como una contraseña.
- El endpoint está atado a una red y un entorno.
- Primero la red de prueba y después la real.

A continuación: capítulo 23. Qué pasa con una solicitud: el gateway, la elección del nodo y tres geografías.

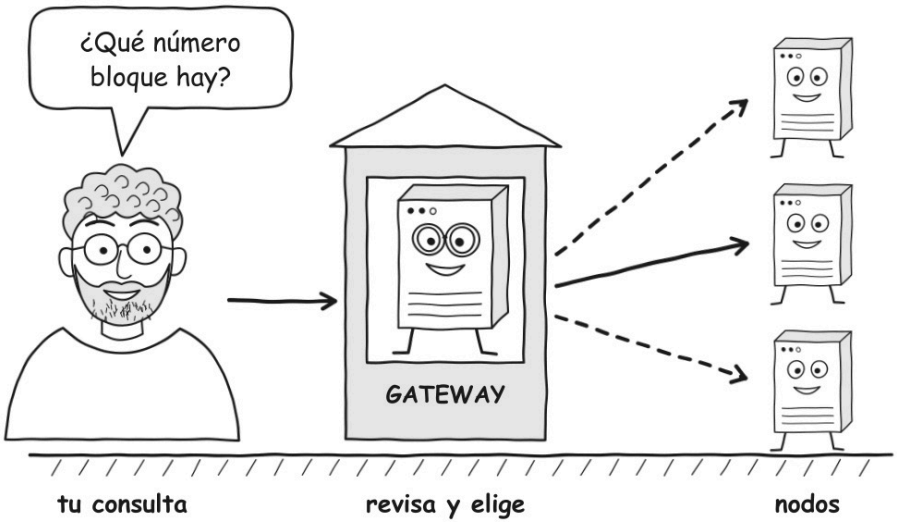
Capítulo 23. Qué pasa con una solicitud

«¿Y quién me ha contestado?»

Artyom envió una solicitud y recibió un número de bloque. Pero ¿adónde fue la pregunta? ¿Qué nodo contestó? ¿Y qué pasa si se rompe en plena campaña de ofertas?

Entendamos: en la entrada hay un gateway

Tu solicitud no llega directamente a un nodo. Primero entra en el **gateway**. Es un programa que hace de despachador: recibe las solicitudes, las comprueba y las pasa al nodo adecuado.



Qué comprueba el gateway por el camino

El orden recuerda al control de entrada de una fábrica. Si no pasas una comprobación, la solicitud no sigue.

Comprobación	Si no la pasa
Si existe el endpoint y la URL es correcta	HTTP 404, `-32000`
Si no está desactivado; si tu IP y tu sitio están permitidos	HTTP 403, `-32000`
Si no preguntas demasiado seguido	HTTP 429, `-32005`
Si esa red tiene ese método	HTTP 404, `-32601`
Si alcanzan los créditos	HTTP 402, `-32006`

Cada red tiene su lista de métodos. Preguntemos algo que no está en ella:

```
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":[]}'
```

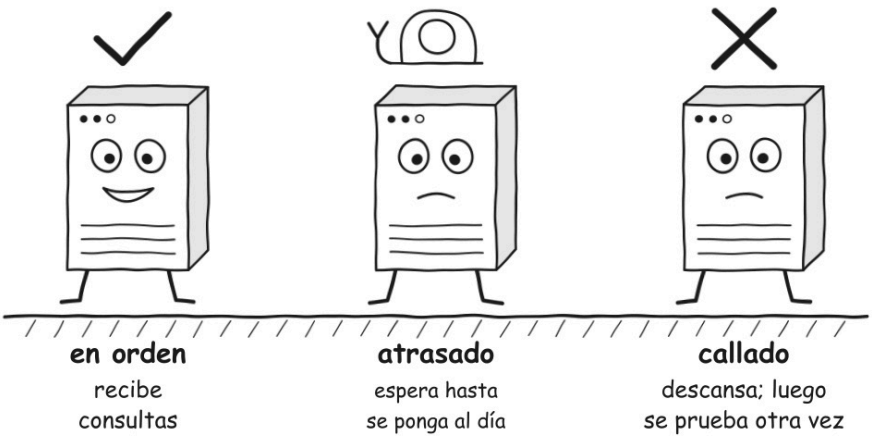
```
{"jsonrpc":"2.0","id":1,"error":{"code":-32601,
"message":"method not found: eth_noSuchMethod"}}  HTTP 404
```

La pregunta no llegó al nodo: contestó el propio gateway.

La elección del nodo: el balanceo

Una red puede tener varios nodos. Repartir las solicitudes entre ellos se llama **balanceo**. Es como las cajas de una tienda: al nuevo comprador lo mandan adonde hay menos cola.

Primero el gateway mira si el nodo está en orden.



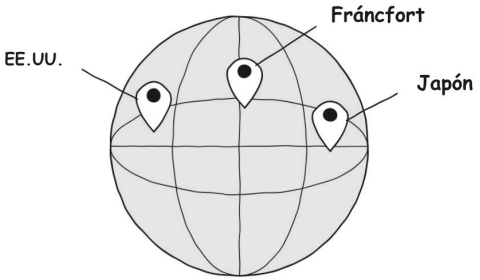
- Un nodo **atrasado** contesta con mucho ánimo, pero con lo de ayer (capítulo 12). Mientras haya nodos al día, las solicitudes van a ellos.

- Entre los nodos sanos, al ocupado le toca menos y al libre, más.
- Si se cortó la conexión o el nodo devolvió un error interno, la misma solicitud pasa al siguiente. Tú no te enteras.
- A un nodo que ha fallado varias veces seguidas se le deja en paz un rato y después se prueba de nuevo.

Hay una salvedad. Si no queda ningún nodo sano, el gateway prueba de todos modos con los atrasados: a veces una respuesta antigua es mejor que ninguna. Por eso, donde la frescura importa, comprueba tú la hora del bloque (capítulo 12).

Tres geografías

RpcNode trabaja en tres geografías: Fráncfort, Estados Unidos y Japón. Cuanto más cerca está el nodo, menos camino de ida y vuelta. El gateway procura tomar el nodo sano más cercano a él y deja los lejanos de reserva.



tres geografías de RpcNode (dibujo, no mapa)

No todas las redes tienen nodos en los tres sitios. Dónde y cómo responde la red que necesitas lo muestra la página `rpcnode.dev/status`.

Probemos: leer las marcas del sobre

Junto con la respuesta llegan las **cabeceras**: anotaciones de servicio, como los sellos de un sobre. Dos de ellas dicen en qué país estaban el gateway y el nodo:

```
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` es Alemania y `JP`, Japón. Esta vez todas las solicitudes las recibió un gateway en Alemania, y para Base y Arbitrum fue hasta Japón. A ti pueden salirte otras letras. Son anotaciones de servicio: no hagas depender de ellas el funcionamiento del sitio.

El error del nodo también es una respuesta

Preguntemos al contrato de USDT: ¿qué pasa si desde una dirección vacía se transfiere una suma enorme? El método `eth_call` solo prueba, sin enviar nada. En el campo `data` van pegados el número de la función de transferencia, el destinatario (`TO`) y la cantidad (`SUM`).

```
TO=000000000000000000000000000000000000000000000000000000000000dead
SUM=0000000000000000000000000000000000000000000000ffffffffffffffff
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
```

Así dice este contrato antiguo «no va a poder ser». El nodo cumplió: informó honradamente de que el programa se detuvo. Otro nodo contestaría lo mismo, así que el gateway te entrega esa respuesta tal cual, con HTTP 200.

Cuando no contestó nadie

También pasa algo peor: el gateway recorrió los nodos y no hay respuesta. Entonces llegan el código `-32002`, el texto `upstream unavailable` («el nodo no está disponible») y un HTTP de la serie 500. Los créditos de esa llamada se devuelven.

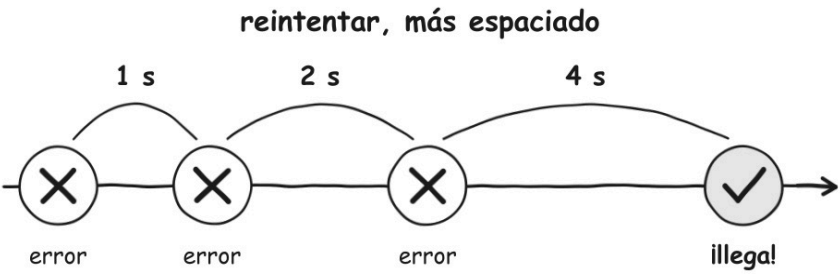
¿Y si es un envío de dinero?

Si el nodo **rechazó** el envío de la transacción, esa es la respuesta. Pero si directamente **no contestó**, el gateway puede pasar la misma transferencia firmada a otro nodo.

El dinero no saldrá dos veces por eso: una transacción firmada tiene un mismo hash y la red la acepta una sola vez. Ahora bien, si quien no recibió respuesta fuiste **tú**, no firmes una transacción nueva al tuntún. Busca primero la antigua por el hash (capítulo 7).

Probemos: reintentar con cabeza

El gateway recorre los nodos, pero la conexión entre tú y el gateway también puede fallar. Reintenta tú, con pausas crecientes: 1, 2, 4, 8 segundos. En inglés se llama **backoff**.



Archivo `ask.js`; se ejecuta con `node ask.js` (hace falta 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('sin conexión')
    }
    await sleep(pause * 1000)
  }
  throw new Error('no se pudo preguntar')
}

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

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

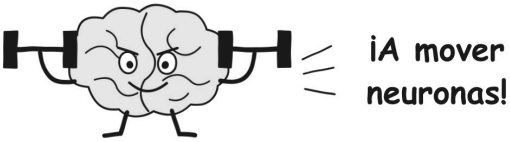
Se reintenta solo lo que puede salir bien a la segunda: «demasiado seguido», un corte, los errores de la serie 500. Un error en la propia solicitud no sirve de nada repetirlo. Y no olvides comprobar el campo `error` después de recibir la respuesta (capítulo 11).

La trampa: «sin respuesta» no significa «sin pago»

Si no se pudo preguntar, el sitio **no sabe** si el cliente pagó. Escribe «estamos comprobando el pago», no «no pagado», y pregunta más tarde.

Y recuerda: un proveedor con varios nodos se cae menos que un solo nodo, pero no «nunca» (capítulo 3).

Para pensar



1. Llegó HTTP 200 y `execution reverted`. ¿Servirá repetir la solicitud cinco veces?
2. Un programa preguntó `eth_blokNumber`. ¿Llegó la solicitud al nodo?
3. Artyom envió una transferencia y no obtuvo respuesta. ¿Qué hacer primero?

Respuestas: 1) no: es una respuesta de verdad, no un fallo. 2) No, el gateway rechazó el método desconocido. 3) Buscar la transacción por el hash.

Lo esencial

- El gateway comprueba la solicitud y elige un nodo sano; si uno no contesta, prueba con otro.
- El error que el propio nodo da sobre tu solicitud te llega tal cual.
- Aun así, tu programa debe reintentar las solicitudes con pausas.

A continuación: capítulo 24. La historia y los datos de archivo.

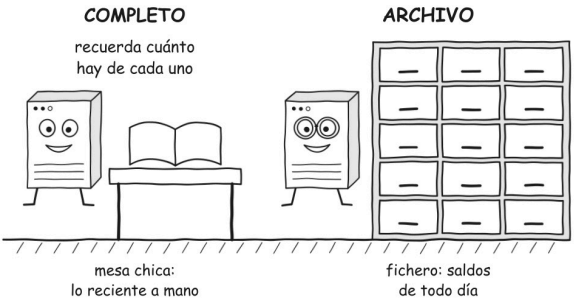
Capítulo 24. La historia y los datos de archivo

«¿Cuánto había el primero de enero?»

El contador le pide a Artyom: «Dígame cuánto había en la dirección de la tienda a principios de año». Sobre hoy, el nodo contesta al instante. ¿Y sobre entonces?

Entendamos: anotaciones y saldos

Recordemos el capítulo 10. **La historia** son los bloques y las transferencias. **El estado**, quién poseía qué en un momento concreto.



Un **nodo completo** tiene a mano los saldos recientes. Uno **de archivo** recuerda los saldos en cualquier bloque, pero a cambio necesita mucho más disco.

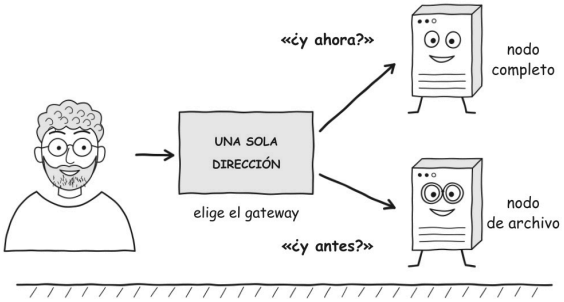
Qué preguntas piden archivo

Pregunta	Ejemplo
Saldo o respuesta de un contrato en un bloque antiguo	`eth_getBalance`, `eth_call` con número de bloque
Eventos de bloques pasados	`eth_getLogs`
«Rebobinar» una transacción paso a paso (trazado)	`trace_…`, `debug_…`
Prueba sobre el estado	`eth_getProof` en un bloque antiguo
Slots antiguos de Solana	`getBlock` con un número antiguo

En cambio, «cuánto hay ahora en la cuenta» y un pago reciente normal no exigen archivo. Una tienda normalmente no lo necesita en el trabajo diario.

Cómo funciona esto en RpcNode

La URL del endpoint es una sola. No tienes que elegir el nodo: el gateway mira la pregunta y decide a quién pasársela.



El archivo está incluido en los planes de pago; en Free no lo hay. Al crear un endpoint en Free, verás en el panel:

> «Los nodos de archivo no están disponibles en el plan gratuito. Tus solicitudes las atienden nodos normales. Responden todo lo que conservan. Lo que solo guarda un nodo de archivo (estado antiguo, historial lejano, algunas trazas) no está disponible: el nodo devuelve su propio error».

- No hay un precio aparte por una pregunta «antigua»: se paga por el método (capítulo 25).
- La profundidad del archivo cambia de una red a otra. Antes de construir sobre él algo importante, prueba tu pregunta más antigua en tu red.
- Las direcciones públicas no atienden preguntas de archivo.

Probemos: el nodo público es estricto

Preguntemos el saldo no «ahora», sino en el número de un bloque de hace apenas diez bloques, unos dos 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
```

Hasta un pasado tan reciente lo considera archivo la dirección pública. Pregúntale con las palabras del capítulo 8: `latest`, `safe` o `finalized`.

Segunda prueba: el nodo pudo haberlo tirado

Un bloque es historia, no estado. Pero tampoco la guardan todos. Desde julio de 2025, los programas de nodos de Ethereum pueden no conservar el contenido de los bloques anteriores al paso al staking (capítulo 10). Pidamos el bloque 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
```

«Historia podada no disponible». El nodo dice honradamente: esto no lo tengo. En un plan de pago, si un nodo normal contesta «esto no lo tengo», el gateway sabe volver a preguntar a uno de archivo, si la red lo tiene.

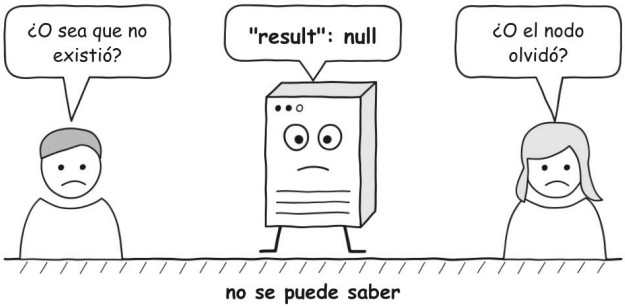
La trampa: «null significa que no existió»

En ese mismo bloque hay una transferencia. Preguntemos por ella con el hash y, a continuación, por un hash que simplemente nos hemos inventado:

```
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}
```

¡Las respuestas son idénticas! La primera transferencia existió de verdad; la segunda no ha existido nunca.



No saques conclusiones de un solo `null`, sobre todo con anotaciones antiguas. A veces significa «no existió», a veces «no lo guardo» y, para una transferencia reciente, «todavía no está en un bloque».

Cuánto recuerda un nodo de Solana

Solana hace bloques varias veces por segundo y las anotaciones se amontonan. Preguntemos el número del slot más reciente y el bloque más antiguo que el nodo todavía conserva:

```
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}
```

La diferencia es de unos 400 000 slots. Al comprobarlo, eso era apenas un día. Y el bloque número 1000 ya hace mucho que se retiró:

```
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}
```

Volvamos al contador

El saldo «a principios de año» es el saldo tras el último bloque del año viejo (contamos en hora universal, UTC). El nodo no te dará de entrada el número de bloque a partir de una fecha. Pero

los bloques van en orden y se puede buscar por **bisección**: tomamos un bloque de en medio, miramos su hora y descartamos la mitad que sobra.

Busquemos el primer bloque de 2026. El script tarda cerca de un 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":'$2'}'
  sleep 2
}
WHEN=1767225600   # 1 de enero de 2026, 00:00 UTC
low=20000000
high=$(( $(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 "sin respuesta"; exit 1; fi
  if [ $(( t )) -lt $WHEN ]; then low=$(( mid + 1 )); else high=$mid; fi
  steps=$(( steps + 1 ))
done
echo "bloque $low, preguntas: $steps"
```

```
bloque 24136053, preguntas: 23
```

Entre millones de bloques, el buscado apareció con 23 preguntas (a ti la cuenta puede variarte en una). Por tanto, el último bloque de 2025 es el 24 136 052 (`0x1704974`). El saldo en él se pregunta igual que en la primera prueba, solo que con este número. La dirección pública lo rechazará: hace falta un endpoint personal en un plan de pago. Para un token, en lugar de `eth_getBalance` se usa `eth_call` con la pregunta `balanceOf` al contrato.

Un tramo largo, por trozos

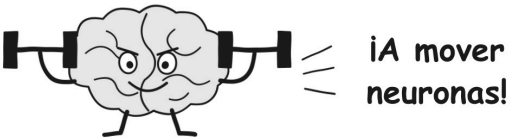
Una solicitud de «todos los eventos del año» lo más probable es que el nodo no la aguante: la respuesta es demasiado grande. El tramo de bloques se corta en partes, se pregunta por turnos y se suman las respuestas.

Esto es un **esbozo**: `rpc` es tu función que pregunta a un endpoint personal, algo como `ask` del 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
}
```

Si no pasa, reduce el trozo; cada uno es una llamada de pago aparte. Y el archivo más barato de la tienda es el **propio**: anota los pagos en la base de datos en cuanto los veas (capítulo 26).

Para pensar



1. El nodo devolvió `null` para el hash de una transferencia antigua. ¿Demuestra eso que el cliente no pagó?
2. Se necesita el saldo de USDC al 1 de julio. ¿Basta con el plan Free?
3. Hay que revisar los eventos de 5000 bloques en trozos de 1000. ¿Cuántas llamadas serán?

Respuestas: 1) no: el nodo podía no guardar esa anotación. 2) El archivo no está incluido en Free, no se puede contar con él. 3) Cinco.

Lo esencial

- Saldos antiguos, eventos antiguos y trazados son trabajo para un nodo de archivo.
- El archivo está en los planes de pago; la URL del endpoint sigue siendo la misma.
- `null` y una respuesta vacía todavía no demuestran que no hubiera nada.

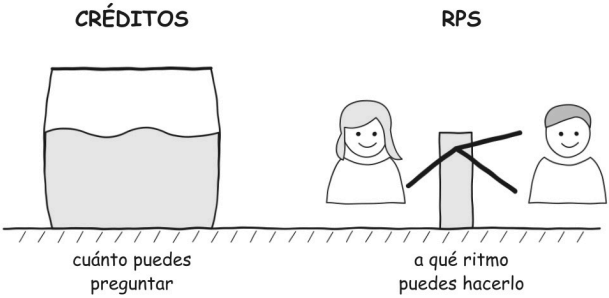
A continuación: capítulo 25. Planes, precio de las llamadas y límites.

Capítulo 25. Planes, precio de las llamadas y límites

«¿Y cuánto saldrá al mes?»

El endpoint funciona. Ahora Artyom quiere saber cuánto le costará la tienda. En la página de precios hay dos cifras principales: créditos y solicitudes por segundo. ¿Qué es cada cosa?

Entendamos: dos medidas distintas



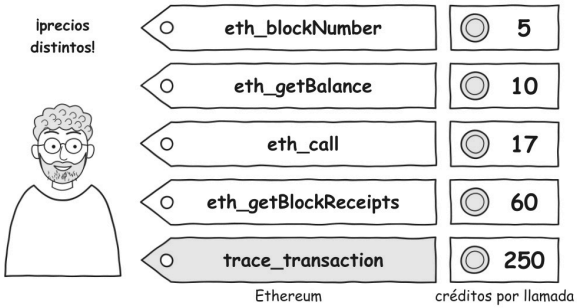
Los **créditos** son el «dinero» interno del servicio: cuánto se puede preguntar en total al mes. El **RPS** son las solicitudes por segundo: con qué rapidez se puede preguntar. Tienen que alcanzar las dos cosas.

Probemos: averiguar el precio de una llamada

Las distintas preguntas cuestan distinto: a una el nodo contesta de memoria; por otra tiene que recorrer un bloque entero. Los precios son públicos y se pueden consultar sin registro. Tomemos algunos métodos de Ethereum (hará falta `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
```



- Un `trace_transaction` cuesta como 50 preguntas del número de bloque.
- Cada red tiene su propia tarifa. Cambia en la dirección `ethereum` por `solana` o `tron` y cambia los nombres de los métodos.
- Un lote (capítulo 11) cuesta la suma de sus llamadas.
- Las suscripciones por WebSocket también son de pago. En Ethereum, abrir una suscripción a los bloques nuevos son 10 créditos, y cada evento enviado, 2.

En el panel los precios están reunidos en la sección «Redes».

Los planes

Datos a 27 de noviembre de 2026. Los vigentes, en `rpcnode.dev/pricing`.

Plan	$ al mes	$ al año	Créditos al mes	RPS	Endpoints
Free	0	—	15 M	15	3
Starter	19	190	50 M	40	5
Build	36	360	100 M	75	12
Grow	99	990	300 M	150	25
Scale	219	2 190	700 M	300	50
Business	449	4 490	1500 M	500	100

Un plan no es solo RPC. Esto es lo que además incluye (capítulos 26 y 27):

Plan	Direcciones vigiladas	Pagos: abiertos / al mes	Transacciones vigiladas	Archivo y todas las redes
Free	—	—	—	no: solo Ethereum y Solana
Starter	15	10 / 200	15	sí
Build	50	25 / 1 000	40	sí
Grow	150	50 / 5 000	80	sí
Scale	400	100 / 20 000	150	sí
Business	1 000	250 / 100 000	400	sí

- Una cuenta nueva puede probar Starter durante 7 días una sola vez, gratis y sin tarjeta.
- Si los planes hechos no te sirven, en la página de precios está «Crea el tuyo».
- Para comprar un plan, pulsa «Comprar» y confirma. Aparece una factura válida por 15 minutos que se puede pagar con criptomonedas.
- Si vence el plazo y no hay un pago nuevo, la cuenta pasa a Free. Ponte un recordatorio.

Hagamos las cuentas de la tienda de Artyom

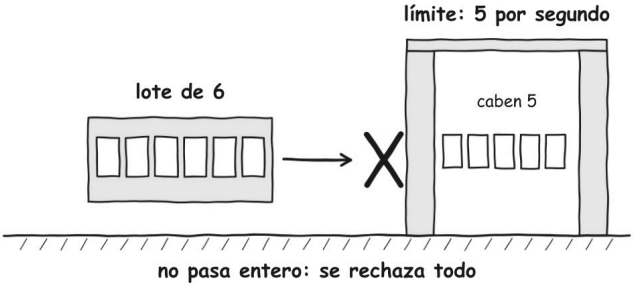
Veinte pedidos esperan el pago, cada uno con su dirección en Ethereum. Tomamos 30 días y suponemos que siempre hay pedidos esperando: es una estimación por arriba.

Cómo pregunta el sitio	Solicitudes	Créditos	Plan suficiente
el saldo de cada dirección cada 10 s	5 184 000	51 840 000	Build
lo mismo cada 30 s	1 728 000	17 280 000	Starter
el número de bloque y un `eth_getLogs` cada 12 s	432 000	4 752 000	Free

La tercera forma es más de diez veces más barata que la primera: una sola pregunta por todas las direcciones a la vez. Pero la comparación tiene una salvedad: las dos primeras formas detectan la llegada de ETH, y la tercera, las transferencias de tokens (capítulo 15). Y en Free solo hay dos redes, Ethereum y Solana.

El límite por segundo

El contador de RPS se pone a cero cada segundo y, según la documentación de RpcNode, cada endpoint tiene el suyo. La solicitud que sobra recibe HTTP 429 y el código `-32005`. **Un lote de N llamadas son N solicitudes**, y pasa entero o no pasa.



Comprobémoslo en la dirección pública, donde el límite es 5. Basta un lote de seis preguntas:

```
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` significa «reintenta dentro de un segundo». La dirección pública protege también los métodos caros:

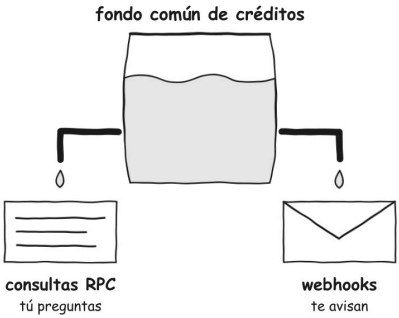
```
curl -s -w '  HTTP %{http_code}\n' https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBlockReceipts","params":["latest"]}'
```

```
{"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
```

Qué se cobra y qué no

Qué pasó	Créditos
El nodo contestó con un resultado	se cobran
El nodo contestó con un error sobre tu solicitud, por ejemplo `execution reverted`	se cobran: también es una respuesta
Rechazo por el límite por segundo	no se cobran
El método no está en la lista de la red	no se cobran
No contestó ningún nodo	se devuelven
Solicitud a la dirección pública	gratis

Una sola reserva para todo



De esa misma reserva se pagan los **webhooks**: mensajes que el servicio envía por sí mismo a tu sitio (capítulo 26). En la red principal de Ethereum, un evento de una dirección vigilada cuesta 450 créditos, y un pago, 750. En las redes de prueba de Ethereum es la mitad.

El saldo restante se consulta en el panel, en la sección «Uso»: allí están también los métodos más frecuentes.

Cuando se acaban los créditos

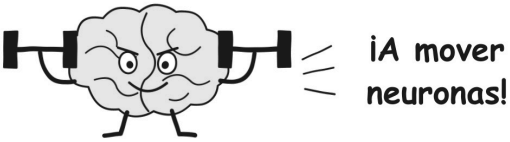
No habrá factura por exceso: no existen recargos por encima del paquete. Pero no cuentes con que el servicio siga funcionando como si nada. Incorpora en el programa el rechazo del capítulo 11 (`-32006`) y vigila el saldo con antelación.

La trampa: elegir el plan por la etiqueta del precio

Primero calcula: **cuántas veces × precio del método**. Unas pocas llamadas pesadas pueden comerse más que miles de ligeras.

Los precios de los métodos y las condiciones pueden cambiar. Según las condiciones del servicio, el pago del plan no se reembolsa, salvo que la ley exija otra cosa, y no se garantiza un funcionamiento sin interrupciones: por una avería grave en un plan de pago se prolonga el periodo pagado. Léelas antes de pagar.

Para pensar



1. El sitio llama a `eth_call` (17 créditos) cada 10 segundos durante todo el mes, 30 días. ¿Cuántos créditos son?
2. En el plan Free se envió un lote de 20 llamadas. ¿Qué pasará?
3. Una solicitud recibió un 429. ¿Se cobraron créditos por ella?

Respuestas: 1) 259 200 × 17 = 4 406 400. 2) Lo rechazan entero: 20 es más que 15. 3) No.

Lo esencial

- Los créditos limitan el volumen del mes; el RPS, la velocidad.
- Cada método tiene su precio; un lote se cuenta por el número de llamadas.
- Preguntar con más cabeza suele salir más a cuenta que tomar un plan mayor.

A continuación: capítulo 26. Eventos por dirección: webhooks y sockets en lugar de preguntar sin parar.

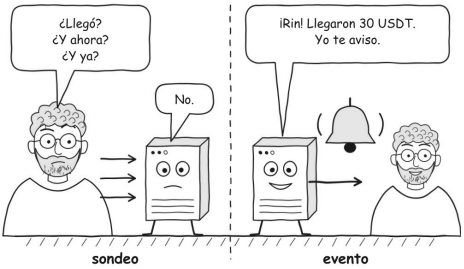
Capítulo 26. Eventos por dirección: webhooks y sockets en lugar de preguntar sin parar

«¿Veinte pedidos, veinte preguntas por segundo?»

En el capítulo 3 echamos la cuenta: veinte pedidos pendientes, una comprobación de cada uno por segundo, y eso ya es más de lo que permiten las 15 solicitudes por segundo del plan Free. Y casi todas las respuestas son iguales: «no ha llegado».

Entendamos: preguntar o esperar el timbre

El sondeo es cuando tu propio sitio pregunta una y otra vez: «¿Llegó?». **Un evento** es cuando te avisan a ti en cuanto algo ocurre.



Probemos: una pregunta por todas las direcciones

Primero preguntemos con más cabeza. No hace falta preguntar por cada pedido por separado. Al método `eth_getLogs` del capítulo 15 se le puede dar de una vez una lista de destinatarios:

```
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"}]}'
```

La pregunta es esta: «¿Hay en el bloque más reciente transferencias de USDT a alguna de estas tres direcciones, de quien sea?». Las direcciones van rellenadas con ceros por la izquierda hasta 64 caracteres. Nos contestaron:

```
{"jsonrpc":"2.0","id":1,"result":[]}
```

Una lista vacía: en este bloque nadie pagó. Las direcciones pueden ser veinte y la solicitud sigue siendo una. Pero habrá que preguntar por **cada** bloque nuevo, contar las confirmaciones por tu cuenta y capturar aparte las transferencias del propio ETH (capítulo 15).

Address Watch: que vigilen por ti

Address Watch, la vigilancia de direcciones, se encarga de ese trabajo. RpcNode analiza los bloques y, cuando una transacción toca tu dirección, avisa a tu servidor.

La dirección se añade en el panel: «Notificaciones» → «Direcciones» → «Añadir dirección». Un programa hace lo mismo con la solicitud `POST /v1/address-watches` y el cuerpo `{"network_slug": "tron", "address": "TU_DIRECCION"}`.

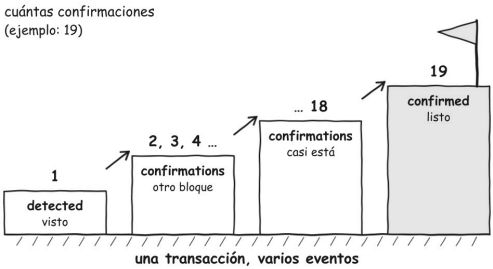
En respuesta llega un `watch_id`, el número de la vigilancia. Una dirección en una red ocupa un **espacio** del plan. Si borras la vigilancia, el espacio queda libre, aunque las altas y bajas también tienen su límite. En Starter hay 15 espacios. En Free no los hay en las redes principales; solo existe una «caja de arena»: unas pocas direcciones en redes de prueba.

Se puede obtener un endpoint para más redes que aquellas que se analizan para eventos. A 27 de noviembre de 2026 son Ethereum, Bitcoin, Solana, TRON, BNB Smart Chain, Base, Arbitrum One, TON, Litecoin, Dogecoin, Bitcoin Cash, Dash, Mantle y Monad, y de las de prueba solo TRON Nile.

La llegada de dinero es un evento de tipo `payment`; la salida, `payout`; todo lo demás (por ejemplo, una autorización approve), `address`. Los bloques nuevos van con el tipo `block`, y sobre `tx` y `payment_order` se habla en el capítulo 27.

Una transacción, varios timbrazos

De una transferencia no se avisa una sola vez, sino a medida que se va «endureciendo» (capítulo 8). Qué ha pasado exactamente lo dice el campo `event`:



event	Significa	Qué debe hacer la tienda
`detected`	la transacción se ha visto en un bloque	mostrar «vemos la transferencia»
`confirmations`	una confirmación más	actualizar el contador
`confirmed`	hay confirmaciones suficientes, éxito	dar el dinero por llegado
`failed`	hay confirmaciones suficientes, la transacción falló	no enviar

Cuántas confirmaciones esperar se define en «Ajustes» → «Confirmaciones»; si no se define, rige el valor de la red.

En Bitcoin, Litecoin y Dash puede llegar antes que nada `pending`: la transferencia sigue en el mempool (capítulo 7). Su tipo es siempre `address`, no `payment`, y no es dinero: esa transferencia puede ser reemplazada.

Webhook: una carta en tu buzón

Un **webhook** es una solicitud que el servicio envía por sí mismo a tu dirección en internet. La URL y las suscripciones por red se configuran en «Notificaciones» → «Ajustes de entrega». La dirección tiene que ser accesible desde internet: `localhost` no sirve.

Si activas el webhook sin elegir nada, llegarán las transferencias entrantes y salientes de todas las redes, pero los bloques no.

Esta es parte de un ejemplo de la documentación: el primer timbrazo sobre 30 USDT en 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 }
```

Las cantidades llegan como cadenas para que nada se redondee; `amount_raw` es lo mismo sin la coma (capítulo 9). Un token que no esté en el catálogo del servicio solo traerá `amount_raw`. En las cabeceras llegan `X-RpcNode-Delivery` (número de entrega), `X-RpcNode-Event` (tipo) y `X-RpcNode-Signature` (firma).

«¿Seguro que son ustedes?»

La URL del webhook está abierta: cualquiera puede llamar y escribir «te han pagado». Por eso cada entrega se firma. Se toman la hora, un punto y el cuerpo, y con ellos se calcula una huella con el secreto del webhook. Ese método se llama **HMAC-SHA256**. El secreto está en el bloque «Secreto e IP»; no es el secreto de la clave de API.

Probemos: firmemos nosotros

El secreto y el cuerpo de aquí son inventados:

```
printf '%s' '1787347206.{"type":"test"}' |
  openssl dgst -sha256 -hmac 'demo-secret' | awk '{print $NF}'
```

```
e17efb57778b0e0c758a9bf2c5705049c4c383d2af3a2bd2783c8b665c151cde
```

Cambia aunque sea un carácter en el cuerpo o en el secreto y saldrá una cadena completamente distinta. Tu servidor calcula lo mismo y lo compara con la cabecera de la forma `t=…,v1=…`.

Este es un **esbozo** de receptor en Node.js. Lo comprobamos con solicitudes hechas por nosotros, pero no con entregas reales: para ellas hace falta una cuenta en el panel. La comprobación de 300 segundos impide que vuelvan a mandar una entrega de ayer.

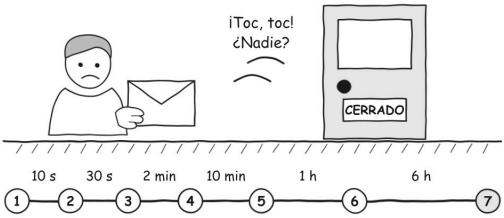
```
import { createServer } from 'node:http'
import { createHmac, timingSafeEqual } from 'node:crypto'

const SECRET = process.env.RPCNODE_WEBHOOK_SECRET
const seen = new Set()      // en una tienda real, una base de datos
createServer(async (req, res) => {
  const parts = []
  for await (const part of req) parts.push(part)
  const body = Buffer.concat(parts)   // los bytes tal como llegaron
  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())  // aquí va tu trabajo
  }
  res.writeHead(200).end()
}).listen(3000)
```

Si no estabas en casa

La entrega tiene éxito si el servidor contesta con un código de la serie 200 en un máximo de 10 segundos. Si no, se repite, y tras el séptimo fracaso se abandona. Por eso:

- Un mismo evento puede llegar dos veces. El número de entrega es el mismo: filtra por él.
- No se promete el orden: `confirmations` puede adelantar a `detected`.
- El trabajo largo hazlo **después** de responder.



pausas cada vez mayores; siete intentos, unas siete horas

Comprueba de antemano tu URL con una entrega de prueba: es gratuita. El registro de entregas del panel guarda las anotaciones durante un día.

Socket: una línea abierta

La segunda forma es **WebSocket**: tu programa abre la conexión y escucha; para eso no necesitas una dirección propia en internet. No es la suscripción `eth_subscribe` del capítulo 11: allí el nodo habla de la red; aquí el servicio habla de tus direcciones y pedidos. El cuerpo es el mismo que el del webhook, pero lo que te perdiste mientras no estabas **no se repite**.

El protocolo es compatible con Pusher (sirve la biblioteca `pusher-js`) y la clave es el número de tu clave de API. El socket se activa aparte, con el interruptor «Entregar por WebSocket». Si activas las dos formas, cada aviso se entrega y se paga dos veces.

Cuánto cuesta

Los eventos gastan los mismos créditos que las solicitudes. Un evento de dirección en la red principal de TRON o Ethereum costaba, a 27 de noviembre de 2026, 450 créditos, **una sola vez por transacción**: las confirmaciones y el resultado son gratis. Se paga por la entrega, aunque tu servidor después no la acepte.

Calculemos un mes de 30 días en Ethereum, donde cada solicitud de la tabla cuesta 17 créditos:

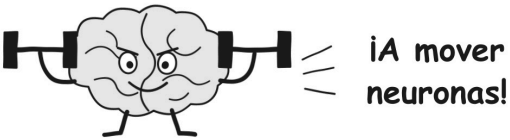
Cómo nos enteramos del pago	Solicitudes o eventos	Créditos
20 pedidos, `eth_call` cada 10 segundos (capítulo 13)	5 184 000	88 128 000
un `eth_getLogs` por cada bloque, cada 12 segundos	216 000	3 672 000
Address Watch, 200 transferencias en el mes	200	90 000

Starter tiene 50 millones de créditos: la primera forma no cabe. Es solo un ejemplo. Cuantas más transferencias, más cuestan los eventos, y las transferencias pequeñas que otros envíen a tu dirección también cuentan.

La trampa: «llegó un evento, así que envío»

- **No todo evento es un pago:** `detected` aún no es definitivo y `payout` es un gasto tuyo.
- **El token se reconoce por la dirección del contrato** del campo `token`, no por el nombre (capítulo 9).
- **Comprueba de antemano las transferencias especiales,** por ejemplo el ETH enviado desde dentro de un contrato (capítulo 15).
- **Los eventos de dirección no son un archivo:** por la API están disponibles durante un día.

Para pensar



1. El servidor de la tienda estuvo apagado una hora. ¿Qué pasa con las entregas del webhook y qué con los avisos del socket?
2. Llegaron dos entregas con el mismo `X-RpcNode-Delivery`. ¿Cuántas veces se abona el dinero?
3. En un mes hay 1000 transferencias en TRON a direcciones vigiladas, con entrega solo por webhook. ¿Cuántos créditos son?

Respuestas: 1) los webhooks se repiten hasta agotar los siete intentos; el socket no repite lo perdido. 2) Una. 3) 450 000.

Lo esencial

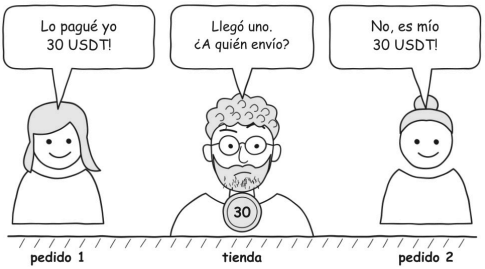
- No preguntes sin parar: que te avisen a ti de la transferencia.
- Comprueba la firma, filtra los duplicados y responde rápido.
- El dinero ha llegado con `confirmed`, no con el primer timbrazo.

A continuación: capítulo 27. Pagos y seguimiento de transacciones.

Capítulo 27. Pagos y seguimiento de transacciones

«El dinero llegó. ¿Y de quién es?»

Artyom recibió un evento: a la dirección han llegado 30 USDT. Abre la lista de pedidos y hay dos de treinta. El evento sabe de la dirección y la cantidad, pero no sabe nada del carrito del cliente.



Entendamos: un pago es una espera

Un **pago** (sección «Pagos» del panel) es una espera anotada de antemano: tal pedido aguarda tanto en tal dirección hasta tal hora. Cuando llega una transferencia que encaja, el propio servicio dice el número de pedido.



La dirección la pones tú. El dinero va del cliente directamente a ti; el servicio no necesita tu frase secreta (capítulo 6).

El pago se crea con el botón «Nuevo pago» o con la solicitud `POST /v1/payments`:

Campo	Qué es
`order_id`	tu número de pedido, sin repetirse
`network_slug` , `address`	la red y tu dirección en ella
`amount`	la cantidad **como cadena**: `"30"` , `"12.5"`
`token_contract`	contrato del token; sin él se espera la moneda de la red
`ttl_seconds`	plazo en segundos; sin él, una hora; como máximo, 30 días

Probemos: llamar sin clave

```
curl https://api.rpcnode.dev/v1/payments
```

```
{"ok":false,"error_code":"public_api.key_missing",
 "error":"Missing X-Api-Key header."}
```

Es una respuesta con el código 401: «¿y usted quién es?». Cada solicitud a esta API se firma con el secreto de la clave de API (sección «Claves de API»). Se firman cuatro líneas: la hora, el método, la ruta y la huella del cuerpo.

Este es un **esbozo** en lenguaje de shell. Comprobamos su sintaxis y el cálculo de la firma, pero no lo enviamos: hace falta una clave personal.

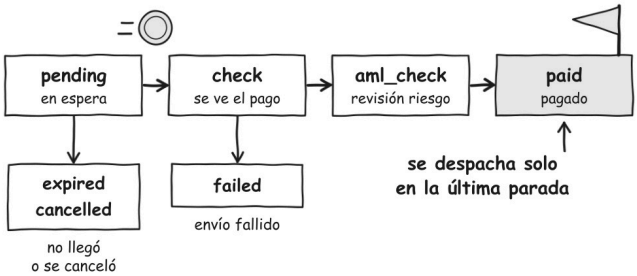
```
BODY='{"order_id":"coffee-1001","network_slug":"tron",
"address":"TU_DIRECCION","amount":"30","ttl_seconds":1800,
"token_contract":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t"}'
TS=$(date +%s)
HASH=$(printf '%s' "$BODY" | openssl dgst -sha256 | awk '{print $NF}')
SIG=$(printf '%s\nPOST\n/v1/payments\n%s' "$TS" "$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"
```

En respuesta llegan el `payment_id` , el estado `pending` y el plazo `expires_at` . Si repites la solicitud con el mismo `order_id` y la misma red, dirección, token y cantidad, volverá el mismo pago. Con otros, el error `payment.order_exists` .

El camino de un pago

Cada paso llega como un evento `payment_order`, por webhook o por socket (capítulo 26).



`event`	Qué pasó
`check`	se ve el dinero, esperamos confirmaciones
`confirmations`	hay más confirmaciones
`aml_check`	las confirmaciones bastan; se está comprobando el riesgo
`paid`	pagado: se puede enviar
`expired`, `cancelled`	venció el plazo o cancelaste un pago pendiente
`failed`	la transacción falló en la red (no lo construyas todo sobre este estado: el pago puede simplemente vencer)

AML es la comprobación de si el dinero enviado está ligado a un mal historial. Donde existe, el objeto `aml` del evento `paid` trae una puntuación `score` de 0 a 100 (cuanto más, más riesgo) y un nivel `risk_level`. No se cobra aparte. No existe en todas las redes: donde no la hay, el pago pasa a `paid` directamente.

Es una pista, no una sentencia ni un dictamen jurídico: qué hacer con una transferencia de riesgo lo decides tú. Si un pago se atasca en `aml_check`, todavía no está pagado; escribe a soporte.

Parte de un ejemplo de la documentación:

```
{ "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" } }
```

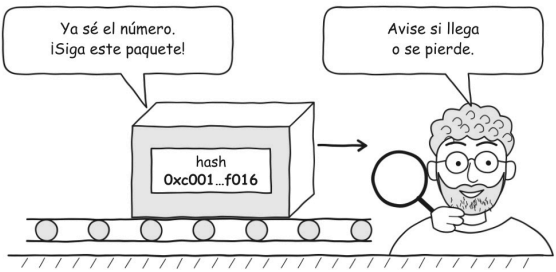
La letra pequeña

- **Menos de lo debido no cuenta.** El pago sigue esperando y vencerá. En la documentación existe el estado `underpaid`, pero no construyas la tienda sobre él: detecta el pago incompleto por el evento de dirección.
- **Más de lo debido sí cuenta.** Cuánto llegó de hecho lo dice `received_amount_raw`.
- **Una transferencia tardía no paga.** La que se vio a tiempo reúne tranquilamente sus confirmaciones también después del plazo.
- **A cada pedido, su dirección.** Así no se confunden dos «de treinta» (capítulo 28).
- **El espacio de la dirección no se libera solo** tras `paid` o `expired`: borra la vigilancia. Tras cancelar un pago, la dirección añadida solo para él se retira sola.

Los pagos están en los planes de pago: en Starter, 10 abiertos a la vez y 200 al mes. Un pago en la red principal de TRON o Ethereum cuesta 750 créditos, y solo se cobran cuando se ve el dinero. En Free hay una caja de arena: 3 pagos abiertos en redes de prueba, ahora mismo TRON Nile. La comprobación de riesgo allí no es real y va marcada con `"sample": true`.

Seguimiento de transacciones

También pasa al revés: ya tienes el hash. Artyom devolvió dinero a un cliente o pagó a su proveedor y quiere saber si llegó.



«Notificaciones» → «Transacciones» → «Añadir hash», o `POST /v1/tx-watches` con el cuerpo `{"network_slug": "tron", "tx_hash": "HASH"}`. El hash se puede añadir antes de que aparezca en la red.

Llegan eventos `tx`: `detected`, después `confirmations` y al final `confirmed` o `failed`. Si el hash nunca se encontró, `cancelled`. Llegan si para esa red están activadas las transferencias entrantes o salientes.

Un seguimiento abierto ocupa un espacio y lo libera al terminar. En Starter hay 15 espacios; en Free no existe este servicio. La transacción se paga una vez: en TRON y Ethereum son 150 créditos.

Probemos: contar las confirmaciones nosotros

Esto es lo que ocurre por dentro en cualquier seguimiento así. Tomemos la transferencia de USDT del 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"}
```

El estado `0x1` es éxito. El bloque de la transferencia cuenta como primera confirmación, así que `echo $((0x19464c5 - 0x194646a + 1))` da **92**. A ti te saldrán más.

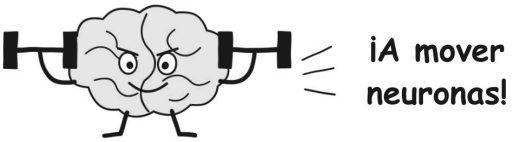
La trampa: dos timbrazos por el mismo dinero

El pago pone la dirección bajo vigilancia. Por eso sobre una misma transferencia pueden llegar tanto `payment` (evento de dirección) como `payment_order` (evento del pago). Cada uno se paga a su precio.

> **Regla**: el pedido solo lo cierra un `payment_order` con `event` = `paid`. Y coteja el número de pedido, el `payment_id`, el token y la cantidad.

Y no preguntes por el pago sin parar: la API tiene su propio límite, 100 solicitudes por minuto por cuenta. Se pregunta después de una caída, para cotejar: `GET /v1/payments/orders/{orderId}`.

Para pensar



1. Un pago de 30 USDT; llegaron 29,5. ¿Qué pasa con el pago?
2. Venció el plazo y la transferencia se vio un minuto después. ¿Dónde está el dinero y qué pasa con el estado?
3. Llegó el evento `check`. ¿Se puede enviar el café?

Respuestas: 1) no está pagado y vencerá a su hora. 2) El dinero está en tu dirección y el pago sigue en `expired`*: lo resuelves a mano. 3) No, espera* `paid`*.*

Lo esencial

- El pago une la transferencia con el pedido; el seguimiento del hash, con la suerte de la transacción.
- Se envía con `paid`, no con el primer timbrazo.
- Los pagos incompletos, los retrasos y los espacios ocupados son cosa tuya.

Fin de la quinta parte. Artyom sabe obtener un endpoint, entiende el camino de una solicitud y los planes, y se entera del dinero sin sondear.

A continuación: parte 6, capítulo 28. Montamos la tienda: factura, dirección, espera del pago y confirmaciones.

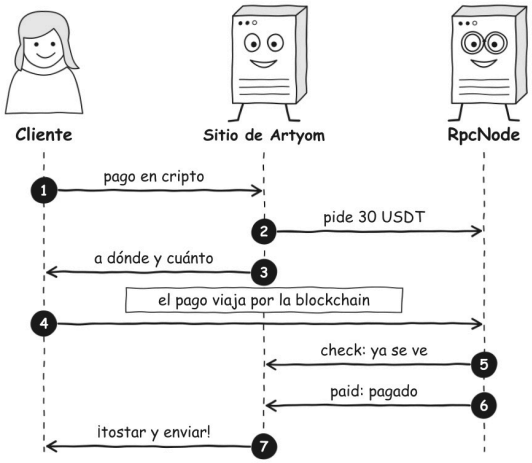
Capítulo 28. Montamos la tienda: factura, dirección, espera del pago y confirmaciones

«Por separado se entiende todo. ¿Y junto?»

Artyom ya sabe de direcciones, confirmaciones, webhooks y pagos. Y entonces un cliente pulsa «Pagar con criptomonedas». ¿Qué debe hacer el sitio a partir de ahí, paso a paso?

Entendamos: quién participa

Son cuatro. El **cliente** con su billetera. El **sitio de la tienda** con su base de datos de pedidos. **RpcNode**, que vigila la red. Y la **billetera de Artyom**, donde están las claves. En la conversación del sitio con el servicio, la billetera no participa.

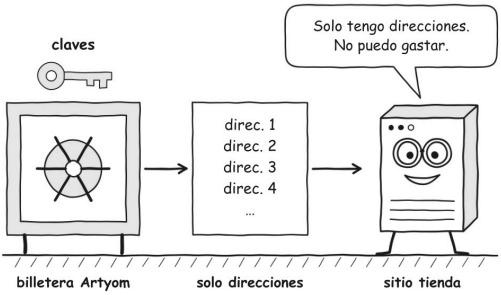


Mira el paso 4. El dinero no va al sitio ni a RpcNode. Cae directamente en la dirección de Artyom, y RpcNode solo lo detecta en un bloque.

De dónde sacar direcciones sin claves en el servidor

En el capítulo 6 prometimos: a cada pedido, su dirección. Y allí mismo dijimos que la clave privada no pinta nada en el servidor de la tienda. ¿Cómo se compagina?

Las direcciones se pueden preparar aparte de las claves.



Hay dos formas.

	Lista de direcciones	Clave pública extendida
Qué hay en el servidor	direcciones ya hechas, una por pedido	una sola clave (a menudo empieza por `xpub`) con la que el programa calcula las direcciones
Dónde sirve	en cualquier red	Bitcoin, Ethereum, TRON y su familia; en Solana y TON no se puede así
Si hackean el servidor	se ven las direcciones de la lista	se ven todas tus direcciones y todos sus ingresos
Trabajo que da	reponerla a tiempo	configurarla una vez

Con una clave así no se puede gastar dinero. Pero el estándar BIP-32 advierte: esta clave **junto con una sola** clave privada de la misma rama permite calcular todas las claves privadas de la rama. Por eso no exportes de una en una las claves de esa billetera. A un principiante le resulta más sencillo empezar con la lista.

Saca la lista de una billetera cuyas claves no estén en un servidor en internet. Es cómoda una **billetera de hardware**: un dispositivo aparte que guarda las claves dentro de sí. Antes de cargar la lista, coteja varias direcciones con su pantalla. Y en TON muestra al cliente la dirección en la forma `UQ` (capítulo 19).

Probemos: la base de datos no deja confundirse

Repartir direcciones es trabajo para una base de datos. Esta es de juguete: dos direcciones libres y tres pedidos. Hace falta el programa `sqlite3`, que muchos sistemas ya traen.

```
sqlite3 :memory: <<'SQL'
CREATE TABLE addresses(address TEXT PRIMARY KEY, order_id TEXT UNIQUE);
INSERT INTO addresses(address) VALUES ('direccion-1'), ('direccion-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 'no quedó dirección para el tercero' WHERE changes() = 0;

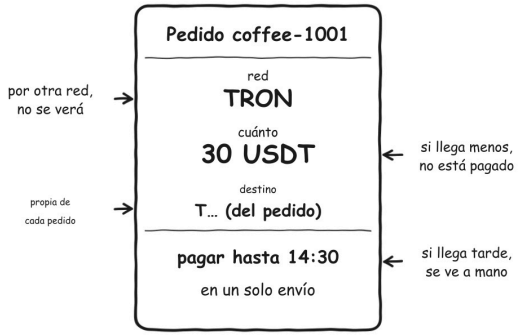
UPDATE addresses SET order_id = 'coffee-1001' WHERE address = 'direccion-2';
SQL
```

```
coffee-1001|direccion-1
coffee-1002|direccion-2
no quedó dirección para el tercero
Runtime error near line 15: UNIQUE constraint failed: addresses.order_id (19)
```

No hay con qué emitir la tercera factura: toca reponer la lista. Y el último comando quería darle al pedido 1001 una segunda dirección, y la base de datos se negó. De eso se ocupan las palabras `PRIMARY KEY` y `UNIQUE`: cada dirección figura una sola vez, y un pedido no tiene más de una.

Y no repartas las direcciones antiguas en una segunda vuelta. Una transferencia atrasada de un pedido anterior pagaría el nuevo.

Emitimos la factura



El cliente necesita cinco cosas: red, moneda, cantidad exacta, dirección y plazo. Y un ruego: que lo transfiera todo **en una sola transferencia**, porque dos mitades no se suman. Adviértele: si la billetera o el exchange descuentan la comisión de la propia cantidad, llegará menos.

Convierte a la moneda, en el momento del pedido, los precios en rublos o dólares, y mantén el cambio hasta que venza el plazo. Por eso el plazo se hace corto, por ejemplo media hora.

Ahora hace falta el pago del capítulo 27. Abajo hay un **esbozo** en Python. En él, la función `rpcnode` envía una solicitud firmada: escríbela según el capítulo 27 o toma el cliente oficial de RpcNode (los hay para Python, TypeScript, PHP, Go y Kotlin). La base de datos es la misma, de dos tablas.

```
USDT = 'TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t'    # contrato USDT en TRON
# addresses(address, order_id)
# orders(id, payment_id, amount_raw, status: al principio 'waiting')

def take_address(db, order_id):
    with db:                                    # una transacción de la base
        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]                      # al repetir, la misma
        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('no quedan direcciones')
        return free[0][0]
```

```
def checkout(db, rpcnode, order_id, amount):   # amount: una cadena
    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        # al cliente: address, amount, expires_at
```

Lo principal aquí es el orden: la dirección se asigna al pedido **antes** de dirigirse al servicio. Si se corta la conexión, la llamada repetida tomará la misma dirección y el mismo número de pedido, y el servicio devolverá el mismo pago. Lo comprobamos con una función `rpcnode` inventada; con el servicio real no ejecutamos el esbozo.

Crea el pago cuando el cliente haya elegido pagar con criptomonedas, no por cada carrito: el límite mensual del plan cuenta los pagos creados, también los que no se pagan.

RpcNode reconoce el pago por la dirección, la moneda y la cantidad. El comentario del capítulo 19 y la marca `reference` del capítulo 18 no intervienen aquí: el pago no tiene un campo así. XRP Ledger y Stellar, del capítulo 20, no están en la lista de redes con pagos (capítulo 26).

La espera del pago

La página del pedido pregunta cada pocos segundos a **tu servidor**, y el servidor contesta desde su base de datos. La base se actualiza con los webhooks. No se puede preguntar a RpcNode directamente desde el navegador: el secreto de la clave acabaría en manos del cliente, y además el límite de la API son 100 solicitudes por minuto por cuenta.

Estado	Qué ve el cliente	Qué hace la tienda
`pending`	«Esperamos la transferencia hasta las 14:30»	nada
`check`	«Vemos la transferencia. Confirmaciones: 3 de 19»	no envía
`aml_check`	«Estamos revisando la transferencia»	espera
`paid`	«Pagado, preparamos el pedido»	envía
`expired`	«Venció el plazo. No transfieras con esta factura»	cierra la factura
`failed`	«La transferencia no se realizó»	propone empezar de nuevo
`cancelled`	«Factura cancelada»	cierra la factura

Mientras la transferencia no entre en un bloque, el pago guarda silencio. Por eso escribe «esperamos», no «no pagado».

Cuántas confirmaciones esperar

Se define en «Ajustes» → «Confirmaciones», por separado para cada red. Si no se define, rige el valor de la red. La cifra necesaria se ve en el pago, en el campo `required_confirmations`.

Poner menos que el valor por defecto es arriesgado: los bloques recientes a veces se sustituyen (capítulo 8). Y la cifra es una por red, no por pedido. Si quieres esperar más por un saco de grano que por una taza, tras `paid` espera un poco más y compruébalo tú, como en el capítulo 27.

Libera espacio

Hay dos tipos de sitios. El **espacio de pago** se libera solo cuando el pago pasa a `paid`, `expired`, `failed` o `cancelled`. El **espacio de dirección** no se libera solo (salvo al cancelar, capítulo 27).

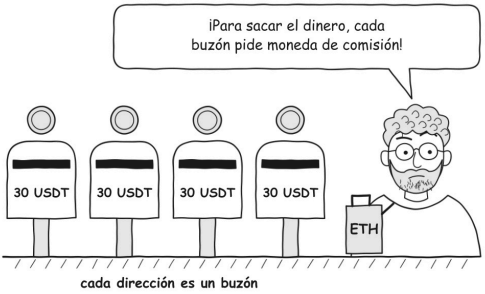
En Starter son 10 pagos abiertos, 15 direcciones vigiladas y 200 pagos al mes. El undécimo pago abierto recibe el rechazo `payment.slot_limit`. Y cuando están ocupadas todas las direcciones, llega `watch.address_limit`.

Deja reposar un poco el pedido cerrado, por si la transferencia se retrasó. Después retira la vigilancia: busca su `watch_id` con la solicitud `GET /v1/address-watches?network_slug=tron&address=…` y envía `DELETE /v1/address-watches/{watch_id}`. A partir de ahí no habrá avisos sobre la dirección, aunque el dinero todavía puede llegar a ella.

Las altas y bajas de direcciones también están limitadas: por día, por semana y por mes, y cuantos más espacios, mayor el margen. Una dirección nueva por pedido va gastando ese margen. Un día de ofertas puedes recibir `watch.quota` incluso con espacios libres. Si esperas muchos pedidos, averigua el tope de antemano.

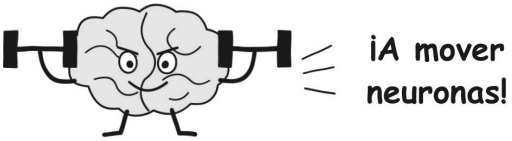
La trampa: el dinero repartido en cien direcciones

Una dirección por pedido es cómoda para la contabilidad, pero luego el dinero queda separado. Para reenviar USDT desde una dirección, en ella misma hace falta moneda para la comisión (capítulos 9 y 17): ETH en Ethereum, TRX o energía en TRON. Cien pedidos son cien transferencias aparte. En Bitcoin, muchos «trozos» pequeños también encarecen el envío (capítulo 14). Haz esa cuenta antes de elegir red para las compras pequeñas.



Y una cosa más. Las billeteras del estándar BIP-44 dejan de buscar tras 20 direcciones vacías seguidas. Si hay muchos pedidos abandonados, puede que la billetera no muestre el pago hecho a una dirección lejana. El dinero está a salvo, pero comprueba que sepa buscar más allá.

Para pensar



1. El cliente recargó la página de pago. ¿Hacen falta una dirección nueva y un pago nuevo?
2. Artyom ya tiene 10 pagos abiertos en Starter. Llega un undécimo cliente. ¿Qué pasará?
3. La página de pago pregunta a RpcNode directamente desde el navegador. Nombra dos males.

Respuestas: 1) no: el mismo número de pedido y la misma dirección. 2) El rechazo `payment.slot_limit`*: pídele que espere o proponle otro medio de pago. 3) El cliente ve el secreto de la clave, y el límite de solicitudes se acaba enseguida.*

Lo esencial

- Las claves se quedan en la billetera y en el sitio solo hay direcciones: un pedido, una dirección, sin repetir.
- El cliente pregunta a tu sitio, y el sitio se entera de las novedades por los webhooks.
- Calcula de antemano los espacios y el gasto de juntar el dinero.

A continuación: capítulo 29. Cuando todo sale mal: errores, reintentos, transacciones atascadas y abonos dobles.

Capítulo 29. Cuando todo sale mal: errores, reintentos, transacciones atascadas y abonos dobles

«¡El café salió dos veces!»

Por la mañana Artyom vio que por un pedido se habían enviado dos paquetes. Y otro cliente escribe: «Ya pagué y ustedes siguen en "esperando"». No son milagros. Pasa cuando el programa se escribió solo para el buen tiempo.

Entendamos: tres tipos de males

Mal	Ejemplo	Remedio
Tu solicitud se quedó sin respuesta	se cortó la conexión al crear un pago	repetir sin hacer daño
El aviso llegó mal	dos veces, desordenado o nunca	un cerrojo en la base de datos y conciliación
La transferencia salió mal	se atascó, llegó tarde o llegó menos	reglas y revisión a mano

Cuándo repetir una solicitud a la API

Las pausas crecientes las vimos en el capítulo 23. Pero sin miedo solo se puede repetir lo **idempotente**: lo hagas las veces que lo hagas, el resultado es como el de una sola. La creación de un pago lo es gracias a `order_id`. Repite con el mismo número y los mismos datos. Firma de nuevo cada reintento: la firma incluye la hora.

Los clientes oficiales de RpcNode no repiten solos un `POST` tras un corte o un error de la serie 500. Así que reintentar la creación del pago es cosa tuya.

Qué hacer con las distintas respuestas de la API:

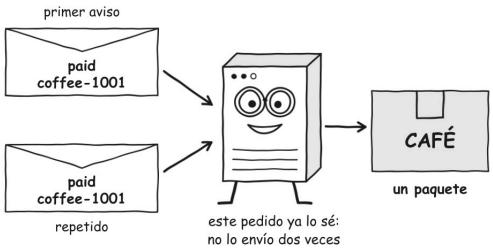
Qué llegó	Qué hacer
nada; 502, 503, 504	reintentar con pausas
429 con la cabecera `Retry-After`	esperar lo que diga
429 `watch.quota` (capítulo 28)	no sirve reintentar enseguida: se agotó el margen
400	error en la solicitud: en `errors` se nombran los campos
401	revisar la clave, la firma y el reloj del servidor
403	el plan, el límite de espacios o la lista de IP permitidas
409 `payment.order_exists`	ese número ya lo ocupa un pago con otros datos: investigar, no cambiar el número al tuntún

Otra cosa es cuando el pago venció y el cliente quiere intentarlo de nuevo. El `order_id` antiguo devolverá el mismo pago vencido. Hace falta un número nuevo, por ejemplo `coffee-1001-2`.

Un aviso, dos veces

Los gemelos salen de todas partes:

- el webhook se repitió porque no contestaste en 10 segundos;
- están activados el webhook y el socket;
- sobre el mismo dinero llegaron `payment` y `payment_order` (capítulo 27);
- el webhook y tu comprobación programada se dispararon a la vez;
- un empleado pulsó «enviar» a mano.



El número de entrega (capítulo 26) filtra las repeticiones de una misma entrega. Pero no salva de un segundo aviso sobre el mismo dinero llegado por otro camino. El cerrojo principal es el estado del pedido: de «esperando» a «pagado» solo se puede pasar una vez.

Probemos: un webhook dos veces

Este es el cerrojo entero. Los avisos nos los hemos inventado. Guarda los dos trozos en un archivo `once.py` y ejecuta `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 'esto todavía no es un pago'
    with db:               # todo lo de dentro es una transacción de la base
        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 'ya contado o no es nuestro'
        db.execute('INSERT INTO to_ship VALUES (?)', (n['order_id'],))
    return 'enviamos'
```

Después, cuatro avisos seguidos y la respuesta del 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))                      # el mismo webhook otra vez
print(4, on_news({**paid, 'event': 'check'}))  # un aviso rezagado
print('paquetes:', db.execute('SELECT count(*) FROM to_ship').fetchone()[0])
```

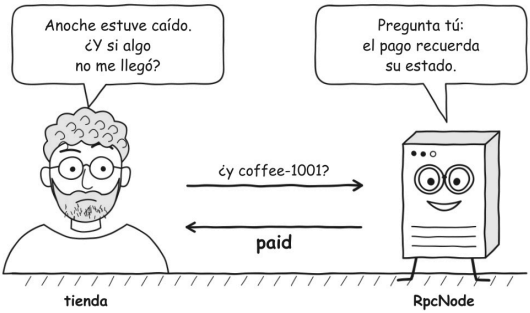
```
1 esto todavía no es un pago
2 enviamos
3 ya contado o no es nuestro
4 esto todavía no es un pago
paquetes: 1
```

Un solo paquete. Todo se sostiene sobre una orden: «pon `paid`, **solo si** el pedido todavía espera y el pago y la cantidad son esos». La marca y la tarea de envío se escriben juntas: o las dos o ninguna. Esto se llama **transacción de la base de datos** (la palabra es la misma que en blockchain, pero la cosa es otra). Probamos también ocho avisos iguales a la vez: salió un solo envío.

En el esbozo del capítulo 26 los números de entrega vivían en memoria y se anotaban antes del trabajo. Con dinero no se puede hacer así: el programa se caerá a la mitad y lo confundirá todo. Primero escribe en la base de datos y después contesta 200.

El aviso no llegó nunca

El webhook se abandona tras siete intentos, que son unas siete horas. El socket no repite nada de lo perdido. Por tanto, parte de las novedades puede extraviarse. El remedio es la **conciliación**: preguntar tú por los pedidos que para ti siguen abiertos.



La solicitud `GET /v1/payments/orders/{orderId}` devuelve el pago. Los nombres allí son un poco distintos: en lugar de `event` mira `status`, la red se llama `network_slug` y el contrato del token, `token_contract`. Arma con la respuesta un «aviso» igual y pásalo por el mismo cerrojo.

Concilia tras una caída del servidor, tras reconectar el socket y de forma programada. Diez pedidos abiertos una vez por minuto son 10 solicitudes de las 100 permitidas.

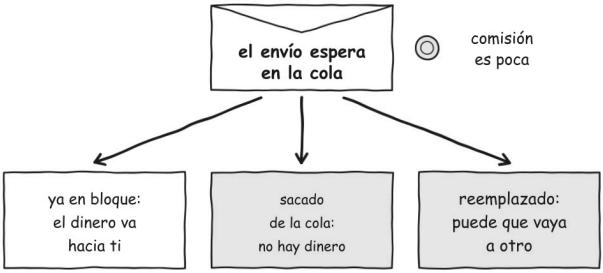
La transferencia se atascó

El cliente envió el hash: «Lo mandé hace una hora». El pago calla. Preguntemos al nodo si conoce esa transacción. El hash de aquí es 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` significa «este nodo no la conoce». Las causas son varias: una errata, otra red, la transferencia aún no ha llegado o ya la descartaron. Y si la transacción apareció pero su `blockNumber` es `null`, está esperando en la cola.



sin entrar en bloque, no tienes el dinero

Red	Qué le pasa a la que espera mucho
Ethereum y similares	retiene todas las transferencias siguientes del remitente (capítulo 15). Se puede reemplazar con una transferencia con el mismo `nonce` y mayor comisión: el programa de nodo geth exige por defecto al menos un 10 % más
Bitcoin	el remitente puede reemplazarla con una transferencia de mayor comisión. El programa Bitcoin Core descarta por defecto de la cola lo que lleva dos semanas
TRON	la transacción tiene fecha de caducidad, por defecto un minuto. Si no llegó a un bloque, ya no entrará
Solana	una transacción normal vale mientras es reciente el bloque al que hace referencia: 150 bloques, apenas unas decenas de segundos

Por regla general, la tienda no puede acelerar una transferencia ajena: hace falta la clave del remitente. Espera el bloque y no envíes. Si mientras tanto el pago venció y aun así la transferencia pasó, el dinero caerá en tu dirección, pero el pago seguirá en `expired`.

¿Se atascó un envío **tuyo**, por ejemplo una devolución? La misma transferencia firmada se puede enviar otra vez: el hash no cambiará. Pero una transferencia nueva es un segundo dinero, salvo que reemplace a la primera. Para ver si una dirección de Ethereum tiene cola:

```
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"}
```

El primer número es cuántas transferencias de la dirección están ya en bloques; el segundo, contando las que esperan. Son iguales: este nodo no ve cola. Es la dirección del remitente del capítulo 15.

Y cuidado con el seguimiento por hash (capítulo 27): la respuesta `cancelled` significa que el servicio no encontró la transacción, no que ya no vaya a aparecer nunca.

El dinero llegó, pero no como debía

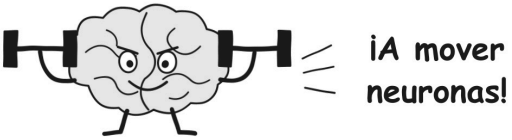
Qué pasó	Qué pasa con el pago
llegó menos	no está pagado y vencerá a su hora; el dinero ya lo tienes
en dos partes	las partes no se suman: lo mismo
llegó más	`paid` ; cuánto de verdad, en `received_amount_raw`
tras `paid` enviaron otra vez	el pago ya no espera: solo se puede saber, si acaso, por el evento de dirección
después del plazo	`expired`
token o red equivocados	no se contabiliza (capítulo 16)
la transacción falló	no hay dinero, no envíes: el pago pasará a `failed` o simplemente vencerá

Decide de antemano qué haces en cada caso y explícaselo al cliente en la página de pago.

La trampa: «lo devuelvo adonde vino»

El cliente pagó de más y Artyom quiere enviar el sobrante a la dirección del remitente. Pero podía ser la dirección común de un exchange: el dinero lo recibirá el exchange, no el cliente. Pregúntale al cliente la dirección y la red para la devolución. Envíalo desde la billetera con tus propias manos, no desde el sitio.

Para pensar



1. No llegó la respuesta a la creación del pago. ¿Se puede repetir? ¿Con qué `order_id` ?
2. El mismo `paid` llegó por webhook y por socket. ¿Cuántos paquetes enviará el programa del ejemplo?
3. En lugar de 30 USDT enviaron dos transferencias de 15. ¿Qué pasa con el pago de 30?

Respuestas: 1) sí, con el mismo y con los mismos datos. 2) Uno. 3) No está pagado: las transferencias no se suman.

Lo esencial

- Repite solo lo que es seguro repetir.
- El envío lo protege un cerrojo en la base de datos, y los avisos perdidos los encuentra la conciliación.
- Mientras la transferencia no esté en un bloque, no hay dinero; para los pagos raros hacen falta reglas de antemano.

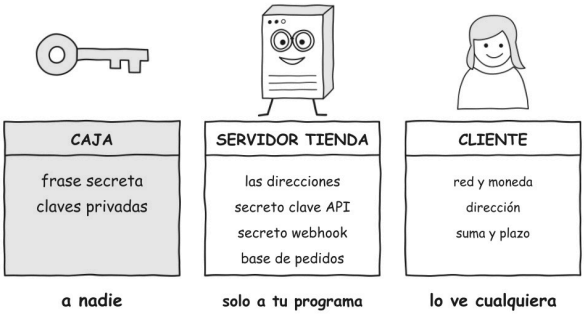
A continuación: capítulo 30. Seguridad y lista de comprobación antes del lanzamiento.

Capítulo 30. Seguridad y lista de comprobación antes del lanzamiento

«Mañana abrimos. ¿Qué se me olvida?»

En la red de prueba todo funciona. Artyom ya escribe a los clientes: «¡Aceptamos cripto!». Pero duerme mal. ¿Qué puede salir mal cuando el dinero sea de verdad?

Entendamos: dónde está cada cosa



El secreto de la clave de API no permite gastar tus monedas: RpcNode no tiene claves privadas. Pero puede causar estragos. Define en la clave las «IP permitidas»: así un secreto robado no funcionará desde una computadora ajena.

La **autenticación de dos factores** es cuando, además de la contraseña, hace falta un código de una aplicación. En el panel está en «Ajustes». Actívala también en el correo: a través del correo a menudo se recupera el acceso a todo lo demás.

Qué pasa si algo de esto se filtra:

Qué se filtró	Qué podrá hacer un extraño	Qué hacer
frase secreta, clave privada	llevarse el dinero sin remedio	transferir cuanto antes el saldo a una billetera nueva
secreto de la clave de API	manejar pagos, vigilancias, endpoints y el webhook, y leer su secreto	borrar la clave, emitir otra, cambiar el secreto del webhook y revisar los ajustes
secreto del webhook	falsificar un aviso `paid`	«Generar un secreto nuevo»
URL del endpoint	gastar tus créditos	crear otra y borrar la antigua (capítulo 22)
acceso al panel	casi todo lo de esta tabla, salvo tus monedas	cambiar la contraseña y activar la autenticación de dos factores

Los secretos no viven en el código

Un secreto metido en el código acaba en el repositorio compartido, en el archivo que se manda a un colaborador externo, en una captura de pantalla. Guarda los secretos en la configuración del servidor (variables de entorno) y cierra a los extraños el archivo que los contiene.

Probemos: busquemos en casa

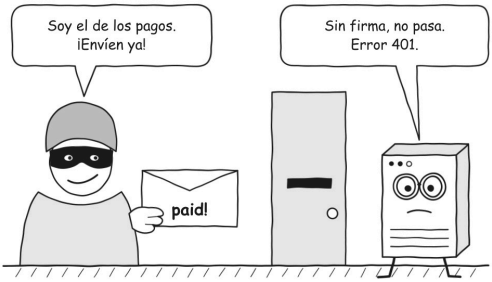
El secreto de la clave de API empieza por `sk_live_`. Hagamos una carpeta de práctica con un error y encontrémoslo:

```
mkdir demo
echo "const SECRET = 'sk_live_ejemplo'" > 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_ejemplo'
```

La búsqueda encontró el primer archivo y no tocó el segundo. Cambia `demo` por la carpeta de tu proyecto. `xprv` es el comienzo de una clave **privada** extendida. Si apareció algo, no basta con borrar la línea: puede haber quedado en el historial de cambios. Da el secreto por filtrado y emite uno nuevo. La frase secreta, en cambio, no se atrapa así, son palabras corrientes: simplemente no la guardes en la computadora.

No te fíes de lo que entra



Probemos nuestra puerta. Guarda el esbozo de receptor del capítulo 26 en `receiver.mjs` y ejecútalo:

```
RPCNODE_WEBHOOK_SECRET=demo-secret node receiver.mjs
```

En otra ventana de la terminal, llama de tres formas ajenas:

```
URL=http://localhost:3000
BODY='{"type":"payment_order","event":"paid"}'
# 1) sin firma
curl -s -o /dev/null -w '%{http_code}\n' $URL -d "$BODY"
# 2) firma al azar
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) firma auténtica, pero de hace una hora
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
```

Tres veces «no pasas». Si en algún caso llegó un 200, la puerta está abierta. Cambia `- 3600` por `- 0` y la tercera llamada recibirá un 200: la firma es reciente y correcta.

Para los pedidos caros, pon un segundo cerrojo. Al recibir `paid`, pregunta tú por el pago (`GET /v1/payments/orders/{orderId}`) y coteja el `payment_id`, la dirección, el token, la cantidad y el estado. Así ni con el secreto del webhook robado se consigue café gratis.

Además, recibe los webhooks por `https` y limita el tamaño de la solicitud entrante: el esbozo del capítulo 26 no lo hace. También se puede dejar entrar solo a las direcciones de la sección «Secreto e IP». Es un añadido a la firma, no un sustituto de ella.

El letrero cambiado

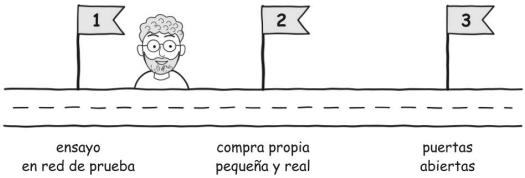


El ladrón no necesita conseguir las claves. Le basta con hackear el sitio y mostrar al cliente su propia dirección. Qué ayuda:

- que desde la administración del sitio no se pueda reemplazar en silencio la lista de direcciones;
- cotejar de vez en cuando las direcciones repartidas con tu billetera, desde otra computadora;
- comprarte a ti de vez en cuando y mirar adónde te piden pagar;
- actualizar los programas del servidor y entrar en la administración con autenticación de dos factores.

Los ladrones también te escriben a ti personalmente: «soporte», «recupere un pago atascado», una transferencia minúscula desde una dirección parecida (capítulo 6). El soporte de verdad no pide la frase secreta.

El ensayo general



Primero, todo en la red de prueba. Los pagos y eventos de prueba funcionan en TRON Nile (capítulo 26). En Free hay una caja de arena: por defecto 3 pagos abiertos y 3 direcciones, y la comprobación de riesgo allí no es real. Las direcciones tampoco se liberan solas en la caja de arena.

Qué representar	Qué debe salir
pagar lo exacto	un paquete
pagar de menos	no se envía nada y el pago vence
enviarte dos veces un mismo aviso	un paquete
apagar tu servidor, pagar y encenderlo	el pago aparece: por la repetición del webhook o por la conciliación
llamar sin firma	401

Después, una compra pequeña y real a ti mismo en la red en la que vayas a vender. De paso, comprueba que tu billetera ve el dinero y que podrás reenviarlo: que alcanza la moneda para la comisión.

Ley e impuestos

En muchos países cobrar en criptomonedas está limitado o paga impuestos. Antes del lanzamiento, revisa las leyes de tu país. Esto no es asesoramiento jurídico. De paso, pregunta a tu contador qué registros de los pagos necesita y guárdalos desde el primer día.

La trampa: «lo comprobé una vez y me olvidé»

La tienda funciona, pero alrededor todo cambia. Venció el plan y la cuenta pasa a Free, donde no hay pagos en la red principal (capítulo 25). Los créditos se gastan también en eventos. La lista de direcciones se está acabando. Ponte recordatorios.

Lista de comprobación antes del lanzamiento

Claves y acceso

- □ la frase secreta está anotada y no está en ningún servidor ni computadora;
- □ en el sitio solo hay direcciones, y varias se han cotejado con la billetera;
- □ no hay secretos en el código; la búsqueda en el proyecto sale vacía;
- □ la clave de API tiene definidas las IP permitidas;
- □ la autenticación de dos factores está activada en el panel, el correo y la administración del sitio.

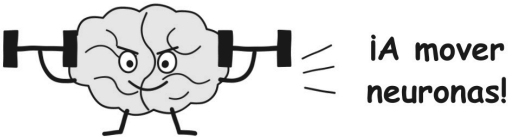
Cobro

- □ un pedido, una dirección; sin repeticiones;
- □ la factura lleva red, moneda, cantidad, dirección, plazo y «en una sola transferencia»;
- □ el contrato del token se tomó de `GET /v1/network-tokens` y se cotejó con el sitio del emisor;
- □ el número de confirmaciones está elegido según tus cantidades;
- □ se comprueba la firma del webhook y un cerrojo en la base de datos protege el envío;
- □ la conciliación de los pedidos abiertos corre de forma programada.

Alrededor

- □ hay reglas para pagos de menos, de más, tardíos y devoluciones;
- □ se vigilan los espacios, los créditos, el plazo del plan y la reserva de direcciones;
- □ está claro con qué se pagará la comisión para juntar el dinero;
- □ se hace copia de la base de pedidos y se ha restaurado desde ella al menos una vez;
- □ el ensayo está hecho y también la compra real propia;
- □ las leyes y los impuestos de tu país están revisados.

Para pensar



1. Se filtró el secreto de la clave de API. ¿Podrán llevarse las monedas de tu billetera? ¿Qué hay que hacer aun así de inmediato?
2. Tu receptor contestó 200 a un mensaje sin firma. ¿A qué te expone?
3. Todas las pruebas en TRON Nile salieron bien. ¿Se puede abrir el cobro en Ethereum sin probar de nuevo?

Respuestas: 1) no, allí no están las claves de la billetera; pero hay que borrar la clave de API, emitir otra, cambiar el secreto del webhook y revisar los ajustes. 2) Cualquiera podrá «pagar» un pedido sin dinero. 3) No: otra red, otras direcciones, comisiones y confirmaciones.

Lo esencial

- Las claves del dinero, lejos del sitio; los secretos, lejos del código.
- No te creas sin más ni un mensaje entrante, ni una captura de pantalla, ni un correo.
- Primero el ensayo, después una pequeña compra real y por último la apertura.

Fin de la sexta parte. Artyom montó la tienda, sabe qué hacer cuando se rompe y cómo protegerla.

A continuación: capítulo 31. DeFi: finanzas sin intermediario.

Capítulo 31. DeFi: finanzas sin intermediario

«¿Y las monedas pueden hacer algo más que estar quietas?»

A Artyom se le ha ido acumulando parte de los ingresos en USDC. Un conocido le aconseja: «Mételo en DeFi, allí el dinero hace dinero solo». ¿Qué lugar es ese, sin ventanilla ni cajero?

> Esto **no es un consejo financiero**, sino una explicación. Los nombres de protocolos son solo ejemplos. En tu país estos servicios pueden estar limitados o pagar impuestos: revisa las leyes.

Entendamos: el banco y el programa

DeFi, las «finanzas descentralizadas», son intercambios, préstamos y otros servicios de dinero que ejecutan contratos inteligentes (capítulo 9). Un conjunto de contratos con reglas comunes se llama **protocolo**.

	Banco	DeFi
Dónde está el dinero	en el banco	en una dirección o en un contrato
Quién ejecuta	los empleados y sus sistemas	un programa en la red
Cuándo funciona	en su horario	las 24 horas
Dónde están las reglas	en el contrato firmado	en el código
Te equivocaste	puedes acudir al banco	normalmente no hay quien lo deshaga
Perdiste el acceso	tu documento de identidad ayuda	sin la clave no se recupera

«Sin intermediario» no significa «sin confianza». Confías en el código, en quienes pueden cambiarlo, en los proveedores de precios y en el emisor de las monedas.

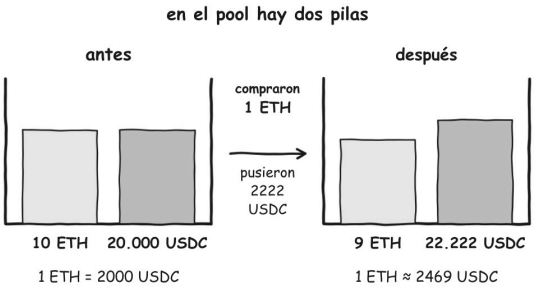
Intercambio sin exchange: el pool de liquidez

En un exchange normal, al comprador le buscan un vendedor. En DeFi a menudo se crea un **pool de liquidez**: un contrato que guarda dos monedas a la vez. Metes una en el pool y sacas la otra. No hay que esperar a una segunda persona.

Cuánto recibirás lo decide una fórmula. En los pools sencillos es esta: $\mathbf{x \times y = k}$, donde x e y son las reservas de las dos monedas. El intercambio no debe reducir ese producto.

El precio aquí es la proporción entre las reservas. Supongamos que en el pool hay 10 ETH y 20 000 USDC. Entonces 1 ETH vale unos 2000 USDC y k = 200 000. Artyom quiere sacar 1 ETH

(de momento no contamos la comisión). Quedarán 9 ETH, así que los USDC deben pasar a ser 200 000 : 9 ≈ 22 222.



Pagará 2222 USDC y no 2000: **su propia compra subió el precio**. Esto se llama **impacto en el precio**. En un pool cien veces mayor (1000 ETH y 2 000 000 USDC), ese mismo 1 ETH habría costado unos 2002 USDC.

Si en el pool el ETH se ha puesto más caro que en otros sitios, alguien llevará allí ETH y sacará USDC hasta que los precios se acerquen. Es el **arbitraje**.

Las monedas las ponen en el pool los **proveedores de liquidez**, que reciben una parte de las comisiones. Pero cuando los precios se separan, el pool va entregando la moneda que se ha encarecido y acumulando la otra. Sin contar las comisiones, si un precio se duplica, el depósito vale aproximadamente un 5,7% menos que si las monedas se hubieran quedado quietas. Esto se llama **pérdida impermanente**: si retiras el depósito en ese momento, se vuelve bien permanente.

Probemos: preguntar a un pool de verdad

Tomemos el pool de USDC y WETH del protocolo Uniswap v2 en Ethereum. **WETH** es ETH en forma de token, uno a uno.

La dirección del pool no se puede sacar de una búsqueda cualquiera. Nosotros tomamos la dirección de la «fábrica» de pools de la documentación de Uniswap, la dirección de USDC de su emisor, Circle, y la dirección de WETH de ethereum.org. Después le hicimos a la fábrica la pregunta `getPair`: «¿dónde está el pool de este par?». La respuesta: `0xB4e16d0168e52d35CaCD2c6185b44281Ec28C9Dc`.

Ahora la pregunta `getReserves`, «¿cuáles son las reservas?». Es un `eth_call` del capítulo 9: solo lee.

```
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"]}'
```

En la noche del 27 de noviembre de 2026 el nodo contestó:

```
{"jsonrpc":"2.0","id":1,"result":"0x000000000000000000000000000000000000000000000000000000079ecc8266bc00000000000000
0000000000000000000000000000000010b762ae141ddef4f50000000000000000000000000000000000000000000000000000000000006b08b4a
f"}
```

Después de `0x` vienen tres números de 64 caracteres:

Número (sin ceros a la izquierda)	Qué es
`79ecc8266bc`	reserva de USDC, 6 decimales: 8 378 617 USDC
`10b762ae141ddef4f50`	reserva de WETH, 18 decimales: 4933,80 WETH
`6b08b4af`	cuándo cambiaron las reservas por última vez, en segundos desde 1970

Qué moneda va primero lo dicen las preguntas `token0` y `token1`: en este pool la primera es USDC. Solo queda dividir:

```
python3 -c 'u = 0x79ecc8266bc / 10**6; w = 0x10b762ae141ddef4f50 / 10**18; print(u, w, u / w)'
```

Se imprimirá `8378617.325244 4933.79553339896 1698.2092728662094`.

Un ETH valía en este pool unos **1698 USDC**. A ti te saldrán otras cifras.

¿Y cuánto darán por una partida grande? La comisión de intercambio aquí es del 0,3%: un 0,25% es para los proveedores y un 0,05% para el protocolo. Con las mismas reservas queda así:

Entregamos	Recibimos en USDC	Por 1 ETH	Peor que el precio 1698,21
1 ETH	1692,77	1692,77	un 0,3%
100 ETH	165 957,86	1659,58	un 2,3%
1000 ETH	1 408 492,58	1408,49	un 17%

El cálculo coincidió hasta la última cifra con las respuestas del contrato auxiliar de Uniswap (router) a la pregunta `getAmountsOut`.

Importante: dividir las reservas vale para los pools del tipo v2. En Uniswap v3 y v4 los depósitos están repartidos por rangos de precios y el precio se calcula de otra manera.

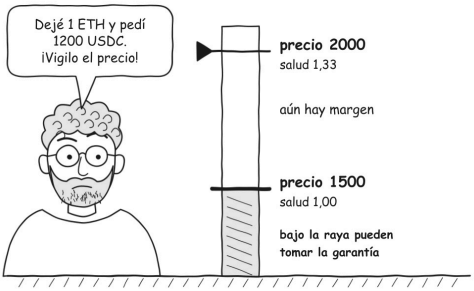
Préstamos con garantía y liquidación

En un protocolo de préstamos (por ejemplo, Aave) se deposita una **garantía** y se pide prestado menos de lo que vale. ¿Para qué? Pongamos que necesitas dólares y no quieres vender tu ETH.

El préstamo tiene una «salud» (health factor):

```
salud = valor de la garantía × umbral de liquidación : deuda
```

Un ejemplo inventado: garantía de 1 ETH a 2000 USDC, umbral del 80% y deuda de 1200 USDC. Salud: 2000 × 0,8 : 1200 ≈ 1,33. Si el precio cae a 1500, quedará exactamente en 1.



Si la salud baja de 1, cualquier participante puede pagar parte de la deuda, y a veces toda, y quedarse por ello con garantía más un recargo. Es la **liquidación**. No habrá ninguna llamada de «traiga más dinero»: todo lo hace el programa, aunque sea en plena noche. La deuda además crece sola, por los intereses.

Stablecoins: en qué se apoya «un dólar»

Capítulo 9: una stablecoin solo **intenta** valer cerca de un dólar. Y se apoyan en cosas distintas. **Reservas en manos del emisor**: dólares y otros activos (USDT, USDC). **Garantía en cripto con margen** guardada en contratos (DAI, USDS; en la garantía puede haber también otras stablecoins). **Un algoritmo** que cambia la cantidad de monedas, sin garantía.

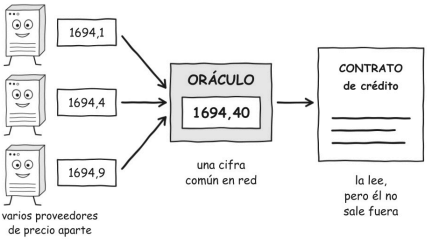
Ningún método es una garantía. Si una moneda así se abarata en un pool, a los proveedores se les acumulará precisamente ella.

El staking en breve

El validador del capítulo 8 necesita 32 ETH. Con una suma menor se participa a través de un **pool de staking común** (no es un pool de intercambio). A cambio suelen dar un token-resguardo. Riesgos: penalizaciones a los validadores, un error en el contrato y la confianza en el operador. Nadie promete cuánto se ganará.

Oráculos: de dónde saca el contrato el precio

El contrato de préstamos necesita el precio del ETH. Pero él no sale a internet: todos los nodos deben obtener el mismo resultado (capítulo 15). Quien escribe el precio en la red es un **oráculo**.



Ejemplo: el oráculo Chainlink. La dirección de su canal «ETH / USD» está tomada del directorio de la propia Chainlink. La pregunta `latestRoundData` pide la anotación más reciente:

```
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":"0x00000000000000000000000000000000000000000000000070000000000008e4f00000000000000000000000000000000000000000000000002773693800000000000000000000000000000000000000000000000006b08aefa000000000000000000000000000000000000000000000000006b08af0f00000000000000000000000000000000000000000000000000000000008e4f"}
```

El segundo de los cinco números, `2773693800`, es el precio con 8 decimales: **1694,40 dólares**. El pool mostraba 1698,21 USDC, una diferencia de en torno al 0,2%. El cuarto número es la hora de la anotación: 25 minutos antes de nuestra pregunta. Este canal se actualiza cuando el precio se aleja un 0,5% del anotado, y además de forma programada, aproximadamente cada hora.

¿Por qué no tomar el precio directamente del pool? Con una operación grande se puede mover por un instante. Pero el oráculo también puede equivocarse o llegar tarde, y la liquidación se hará según su número.

Cómo lee la red una aplicación DeFi

El sitio de intercambios o préstamos es un escaparate. Los datos se los pide a un nodo:

- `eth_call`: lo que hay ahora, es decir, reservas del pool, saldos, precio del oráculo, salud del préstamo.
- **Eventos** (capítulo 15): lo que ha cambiado. El pool v2 anuncia tras cada cambio un evento `Sync` con las nuevas reservas. Se buscan con `eth_getLogs` o se escuchan por WebSocket

(capítulo 11).

- **Envío**: un intercambio cambia el cuaderno. Lo firma la billetera, el nodo lo reenvía (capítulo 7) y cuesta gas.

Esta es la pregunta por los eventos `Sync` de nuestro pool en el bloque más reciente:

```
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":[]}
```

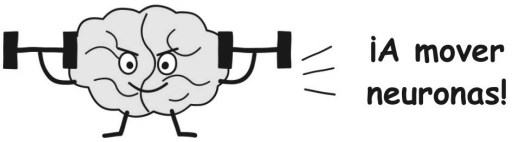
Una lista vacía también es una respuesta: en este bloque el pool no cambió. El tercer número de `getReserves` lo confirma: el último cambio fue unos 10 minutos antes de nuestra pregunta. Los cambios pasados, por ejemplo para un gráfico del precio, ya son archivo o tu propia base de datos (capítulo 10).

La trampa: lo que puede salir mal

Peligro	Qué ocurre	Qué reduce el riesgo
Error en el código	roban el dinero o se queda atrapado	auditoría del código, aunque no es una garantía
Token falso	el nombre suena, la dirección es otra	cotejar la dirección (capítulo 9)
Sin límite (`approve`)	se llevan los tokens más tarde	autorizar poco y poner a cero lo que sobre
Deslizamiento	recibiste menos de lo esperado	fijar la cantidad mínima
Liquidación	la garantía se fue con recargo	mucho margen y vigilancia
Sitio gemelo	una copia del sitio pide una firma	comprobar la dirección; no escribir la frase secreta
«Ganancia segura»	prometen y desaparecen	no participar

El **deslizamiento** es la diferencia entre el precio esperado y el real. Mientras el intercambio espera en la cola, las operaciones de otros mueven el precio, a veces a propósito. La cantidad mínima protege: por debajo de ella el intercambio no se hace, aunque el gas se gaste.

Para pensar



1. En un pool hay 100 ETH y 200 000 USDC. Alguien sacó 20 ETH. ¿Cuántos USDC metió (sin comisión)?
2. Garantía de 2 ETH a 2000 USDC, umbral del 80% y deuda de 2400 USDC. ¿Cuál es la salud? ¿A qué precio del ETH quedará en 1?
3. Un sitio promete un 3% al día sin riesgo y pide autorización para gastar tus USDT sin límite. ¿Qué haces?

Respuestas: 1) quedan 80 ETH, así que los USDC pasan a ser 250 000: metió 50 000, a 2500 por ETH. 2) Cerca de 1,33; a un precio de 1500. 3) No firmar nada y cerrar el sitio.

Lo esencial

- En DeFi los servicios los ejecutan contratos, y el riesgo y la responsabilidad recaen en buena parte sobre ti.
- En un pool del tipo v2 el precio sale de las dos reservas, y una operación grande lo mueve.
- La aplicación lee la red con las mismas consultas que tú: `eth_call` y eventos.

A continuación: conclusión. Del mensaje de un cliente a tu propia tienda.

Comprobado el 27 de noviembre 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.

Conclusión. Del mensaje de un cliente a tu propia tienda

Todo empezó con una línea en un mensaje: «¿Puedo pagar con criptomonedas?». Artyom no sabía entonces qué era un nodo y revisaba la billetera a mano. Mira el camino que has recorrido con él.

Qué sabe hacer ahora Artyom

Parte	Qué aprendimos
1. Por qué hace falta un proveedor	A calcular el gasto en servidores propios y a elegir entre mantener un nodo uno mismo o contratarlo a un proveedor
2. Blockchain desde cero	A entender los bloques, las claves, las firmas, la comisión, las confirmaciones y los tokens
3. Nodos y RPC	A hacer preguntas a un nodo, distinguir los tipos de nodos y comprobar si se ha atrasado
4. Cómo funcionan las redes	A ver en qué se diferencian las redes y elegir la que convenga a los clientes
5. RpcNode	A obtener un endpoint, calcular el precio de las llamadas y enterarse de los pagos sin sondear
6. Práctica	A montar el cobro, sobrevivir a los fallos y proteger la tienda
7. Y ahora qué: DeFi	A leer un pool directamente desde la red y valorar los riesgos con honradez

Siete reglas para recordar

1. La dirección se muestra a todos; la clave privada y la frase secreta, a nadie.
2. Se comprueba todo a la vez: la red, la dirección del contrato del token, el destinatario y la cantidad.
3. El dinero ha llegado cuando la transacción tuvo éxito y reunió sus confirmaciones, no cuando llegó una captura de pantalla.
4. El código 200 todavía no es éxito: lee lo que hay dentro de la respuesta.
5. Un pago, un envío, por muchas veces que llegue el aviso.
6. Cuida el endpoint y los secretos como contraseñas.
7. No te lo creas sin más: pregunta a la red y comprueba la respuesta.

Por dónde empezar mañana

1. Repite la consulta del capítulo 3 al nodo público.
2. Crea tu propio endpoint y una red de prueba (capítulo 22).

3. Ensaya un pago a ti mismo con dinero de juguete (capítulos 28 y 30).
4. Repasa la lista de comprobación del capítulo 30.
5. Revisa las leyes y los impuestos de tu país.

Si algo se ha quedado antiguo

El libro describe las redes y el servicio tal como eran el 27 de noviembre de 2026. Las redes se actualizan y los precios y planes cambian. Si la respuesta de tu nodo no coincide con el libro, cree al nodo y a la documentación reciente. Los planes y las funciones de RpcNode se consultan en el sitio rpcnode.dev.

Hasta la vista

Artyom todavía se está pensando si meter parte de los ingresos en DeFi. En cambio, sabe qué preguntas hacer. Puede que eso sea lo principal que ha sacado de toda esta historia.

Parte de las cifras de este libro quedará anticuada, pero la costumbre de comprobar, no. ¡Mucha suerte a ti y a tu negocio!



Apéndice A. Glosario

Entre paréntesis, el capítulo donde la palabra se explica con más detalle.

AML: comprobación de si el dinero está ligado a un mal historial (27).

API: ventanilla por la que un programa se dirige a otro (11).

Balanceo: reparto de las solicitudes entre varios nodos (23).

Billetera: programa o dispositivo donde se guardan las claves (6).

Blockchain: lista común de registros que se lleva entre muchos con reglas compartidas (4).

Bloque: «página del cuaderno», con transacciones, número, hora y el sello del bloque anterior (5).

Chain ID: número de una red EVM (16).

Clave privada: aquello con lo que se manda sobre el dinero; no se enseña a nadie (6).

Comisión: pago a la red por un sitio en el bloque (7).

Confirmaciones: bloques apilados sobre tu transacción, contando el suyo propio (5).

Consenso: reglas con las que los participantes eligen una sola cadena (8).

Contrato inteligente: programa escrito en la blockchain (9).

Créditos: unidades con las que el servicio cuenta el consumo de solicitudes (3, 25).

DeFi: servicios de dinero que ejecutan contratos inteligentes (31).

Dirección: el «número del buzón» al que envían dinero (6).

Doble gasto: intento de pagar dos veces con la misma moneda (4).

Endpoint: dirección a la que un programa hace preguntas al nodo (3).

Evento (log): anuncio de un contrato sobre lo que ha ocurrido (15).

EVM: máquina virtual de Ethereum que ejecuta los contratos (15).

Finalidad: momento a partir del cual un bloque ya no se sustituye (8).

Firma: prueba de que la transferencia la autorizó el dueño de la clave (6).

Frase secreta: 12 o 24 palabras con las que se recuperan las claves (6).

Gas: medida del trabajo por el que se cobra la comisión en Ethereum y su familia (15).

gRPC: forma de comunicación entre programas con mensajes compactos y un esquema estricto (11).

Hash, «sello»: huella corta de cualquier dato (5).

Idempotencia: se repita las veces que se repita, el resultado es como el de una sola (29).

Indexador: programa que reparte los registros de la blockchain en un «fichero» (10).

L2, «segundo piso»: red que ejecuta las transferencias en su casa y envía a Ethereum las anotaciones sobre ellas (16).

Límite de solicitudes (RPS): cuántas veces por segundo se puede preguntar (3, 25).

Liquidación: venta de la garantía cuando el préstamo deja de estar cubierto por ella (31).

Mempool: cola de transferencias que esperan un bloque (7).

Minería: resolver un problema a base de probar por el derecho a añadir un bloque (8).

Nodo: computadora con el programa de la red: guarda, comprueba y responde (1).

Nodo de archivo: recuerda el estado en cualquier bloque del pasado (10).

Nodo podado: lo comprobó todo y borró los bloques antiguos (10).

Nonce: número de orden de una transacción de la cuenta (15).

Oráculo: quien escribe en la red datos del exterior, por ejemplo un precio (31).

Pool de liquidez: contrato con dos monedas en el que se intercambian según una fórmula (31).

Proveedor: servicio que mantiene nodos y da acceso a ellos (1, 3).

Puente: forma de llevar valor de una red a otra (16).

Red de prueba: campo de entrenamiento con monedas sin precio (22).

Reorganización: sustitución de parte de los bloques recientes por otra rama (8).

REST: forma de preguntar en la que cada pregunta tiene su ruta en la dirección (11).

RPC, JSON-RPC: forma de pedir a un programa de otra computadora que haga algo (11).

Sincronización: cómo alcanza el nodo a la red (12).

Stablecoin: token que intenta mantenerse cerca de un dólar (9).

Staking: participación en la protección de la red dejando monedas en garantía (8).

Token: anotaciones que lleva un contrato inteligente (9).

UTXO: «trozo de dinero» sin gastar en Bitcoin y su familia (14).

Validador: participante que deja una garantía y participa en la creación de bloques (8).

Webhook: solicitud que el servicio envía por sí mismo a tu dirección cuando algo ocurre (26).

WebSocket: conexión que no se cierra, para recibir novedades (11).

Wei: la fracción más pequeña de un ETH (1).

Apéndice B. Las redes del libro en una página

Datos de los capítulos 6, 8 y 14–20, a 27 de noviembre de 2026. Cambian con las actualizaciones de las redes.

Bitcoin y sus parientes (capítulo 14)

Red	Moneda	Un bloque aprox.	La dirección empieza por
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 y las redes EVM (capítulos 15 y 16)

En todas, la dirección empieza por `0x`.

Red	Chain ID	La comisión se paga en	Un bloque 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

Las demás (capítulos 17–20)

Red	Moneda	La dirección empieza por	Definitivo
TRON	TRX	`T`	cerca de un minuto
Solana	SOL	sin comienzo especial	fracciones de segundo
TON	Gram	`EQ`, `UQ`	cerca de un segundo
Sui	SUI	`0x`	menos de un segundo
Aptos	APT	`0x`	menos de un segundo
XRP Ledger	XRP	`r`	3–5 segundos
Stellar	XLM	`G`	unos 5 segundos
Cardano	ADA	`addr1`	poco a poco
Polkadot Hub	DOT	`1`	por la marca
Cosmos Hub	ATOM	`cosmos1`	rápido

Apéndice C. Guía rápida de consultas

Todos los ejemplos funcionan en los nodos públicos de RpcNode: sin registro ni clave, con límite de frecuencia. Para tu proyecto, cambia la dirección por un endpoint personal.

JSON-RPC: un comando para todos los casos

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

Pon el método y los parámetros de las tablas. Para otra red, cambia `ethereum` en la dirección.

Ethereum y las redes EVM

Qué averiguar	Método	Parámetros
número del último bloque	`eth_blockNumber`	vacío
número de la red	`eth_chainId`	vacío
saldo de la moneda	`eth_getBalance`	`"0xDIRECCION","latest"`
recibo de una transacción	`eth_getTransactionReceipt`	`"0xHASH"`
precio del gas	`eth_gasPrice`	vacío
preguntar a un contrato	`eth_call`	`{"to":"0x…","data":"0x…"},"latest"`

Bitcoin y Solana

Dirección	Método	Qué averiguar
`public-bitcoin-mainnet`	`getblockcount`	altura de la cadena
`public-bitcoin-mainnet`	`getblockchaininfo`	estado del nodo
`public-solana-mainnet`	`getSlot`	número de slot
`public-solana-mainnet`	`getBalance`	saldo en lamports; el parámetro es la dirección

TRON y TON: la pregunta va en la propia dirección

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

```
curl https://public-ton-mainnet.rpcnode.dev/api/v2/getMasterchainInfo
```

Códigos de error

Código	Qué pasó
`-32700`	no se puede interpretar la solicitud
`-32601`	método no encontrado o no disponible
`-32602`	parámetros no válidos
`-32005`, HTTP 429	demasiado seguido
`-32006`, HTTP 402	no alcanzan los créditos

Documentación completa y planes vigentes

https://rpcnode.dev

Estamos en Telegram

@rpcnode_dev

Autor del libro: rpcnode.dev

Fin del libro.

En palabras sencillas

Blockchain sin niebla

¿Qué enseña este libro?

Un cliente pregunta: «¿Puedo pagar con cripto?» Qué contestar, cómo saber que el dinero llegó y no perderlo.

Dentro: bloques, claves, comisiones, tokens, nodos y sus consultas, Bitcoin, Ethereum, TRON, Solana, TON y otras redes. Luego práctica: endpoint en RpcNode, pagos, webhooks y defensa de la tienda. Al final, una charla honesta sobre DeFi.

> «Mi sitio sabe sumar el carrito y enviar correos. ¿Cómo sabrá qué pasa en la blockchain?»
>
> **— Artyom,** protagonista



¿Qué tiene de especial?

No hace falta saber nada antes. Dudas de Artyom, respuestas claras, dibujos y ejercicios. Las consultas puedes lanzarlas tú y ver la respuesta real.

rpcnode.dev

docs.rpcnode.dev

Telegram: @rpcnode_dev

Autor: rpcnode.dev