안개 걷힌 블록체인

첫 트랜잭션부터
RpcNode로 만드는 나만의 서비스까지



쉬운 말로 · 그림과 예제와 함께

쉬운 말로 풀어 쓴

안개 걷힌 블록체인

첫 트랜잭션부터 RpcNode로 만드는 나만의 서비스까지



이런 분을 위한 책이에요

- '블록체인'이라는 말은 들어 봤지만 어떻게 돌아가는지는 모르는 분
- 내 웹사이트에서 암호화폐 결제를 받고 싶은 분
- 프로그램은 짜 봤지만 노드에 말을 걸어 본 적은 없는 분

필요한 것

- 호기심. 미리 알아야 할 것은 없어요
- 예제를 따라 해 볼 터미널 있는 컴퓨터
- 첫 요청들은 가입도 키도 없이 됩니다

각 장을 읽는 법

주인공의 질문 **알아보기** **직접 해보기** **함정** **생각해 보기**

지은이: rpcnode.dev · 2026

차례

파트 1. 제공업체가 필요한 이유

1. 크립토 결제를 받고 싶은데, 무엇이 필요할까? 4
2. 전부 직접 하면 얼마나 들까? 9
3. 제공업체는 무엇을 해 주고, 내 노드는 언제 필요할까 15

파트 2. 처음부터 배우는 블록체인

4. 중개자 없는 돈과 기록 21
5. 블록과 체인: 옛 기록을 몰래 고칠 수 없는 이유 26
6. 키, 주소, 서명 31
7. 트랜잭션의 여정: '보내기' 버튼에서 블록까지 36
8. 네트워크는 어떻게 합의할까 41
9. 코인, 토큰, 스마트 컨트랙트 47

파트 3. 노드와 RPC

10. 노드의 종류 53
11. 노드와 대화하는 법 58
12. 내 노드의 속사정: 설치, 동기화, 고장 64
13. 제공업체 고르는 법 70

파트 4. 네트워크는 어떻게 동작할까

14. Bitcoin과 그 친척들 75
15. Ethereum: 계정, 가스, 이벤트 80
16. Ethereum의 친척들과 '2층' 85
17. TRON: 가스 대신 에너지, 스테이블코인이 즐겨 다니는 길 90
18. Solana: 모든 것이 계정에 나뉘어 있다 96
19. TON: 전화 대신 편지 102
20. 다른 접근법: Sui와 Aptos, XRP Ledger와 Stellar, Cardano, Polkadot, Cosmos 107
21. 어떤 네트워크를 고를까 114

파트 5. RpcNode

22. RpcNode에서의 첫 5분 120
23. 요청에는 무슨 일이 벌어질까 126
24. 기록과 아카이브 데이터 132
25. 요금제, 호출 가격, 한도 138
26. 주소 이벤트: 끝없이 묻는 대신 웹훅과 소켓 144
27. 결제와 트랜잭션 추적 150

파트 6. 실습

28. 가게 조립하기: 청구서, 주소, 결제 기다리기, 컨펌 156
29. 모든 것이 꼬일 때: 오류, 재시도, 멈춘 트랜잭션, 이중 입금 처리 163
30. 보안과 출시 전 체크리스트 170

파트 7. 다음 단계

31. DeFi: 중개자 없는 금융 176

책의 마지막

맺음말 183
부록 A — 용어 사전 185
부록 B — 한 페이지로 보는 네트워크 187
부록 C — 요청 치트 시트 189

제1장. 크립토 결제를 받고 싶은데, 무엇이 필요할까?

모든 것은 손님 한 명에게서 시작됐어요

아르티옴은 작은 온라인 가게를 해요. 커피를 볶아서 택배로 보내죠. 카드, 계좌 이체, 착불 결제까지 모두 잘 돌아가고 있어요. 그런데 어느 날 메일 한 통이 옵니다.

> "안녕하세요! 2kg 주문하고 싶은데요, 암호화폐로 결제해도 될까요?"

아르티옴은 마다할 이유가 없어요. 휴대폰에 지갑을 설치하고, 주소를 받아서 손님에게 보냅니다. 몇 분 뒤 화면에 돈이 들어온 게 보여요.

그런데 열 번째 주문쯤 되니 지갑을 손으로 확인하는 일이 지겨워져요. 주문이 결제됐는지 사이트가 알아서 알아차리면 좋겠어요.



알아보기: 노드란 무엇일까

블록체인에 관해 무언가 알고 싶다면, 그것을 보관하고 있는 누군가에게 물어봐야 해요. 그 누군가를 노드라고 부릅니다.

누가 누구에게 얼마를 보냈는지 적어 두는 공용 장부를 떠올려 보세요. 이 장부는 금고 하나에 들어 있지 않아요. 전 세계의 수많은 참가자가 사본을 가지고 있고, 끊임없이 서로 맞춰 봅니다. 이것이 바로 블록체인이에요. 자세한 이야기는 5장에서 할게요.

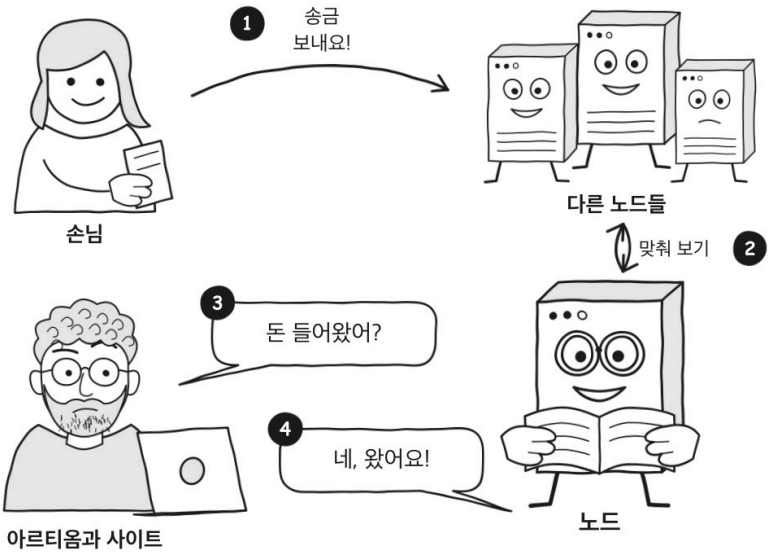
노드는 네트워크 프로그램이 돌아가는 컴퓨터예요. 하는 일은 네 가지입니다.

1. 자기만의 기록 사본을 보관해요.
2. 다른 노드에게서 새 기록을 받아 네트워크 규칙에 맞는지 검사해요.
3. 자기에게 들어온 송금을 다른 노드에게 전달해요.

4. “잔액이 얼마야?”, “그 송금 처리됐어?” 같은 프로그램의 질문에 답해요.

노드에도 여러 종류가 있어서, 어떤 노드는 전체 기록을 다 보관하지 않아요. 이 이야기는 10장에서 합니다.

아르티옴의 사이트는 블록체인을 직접 들여다보지 않아요. 노드에게 질문하고, 노드가 대답합니다.



직접 해보기: 노드에게 던지는 첫 질문

프로그램은 간단한 규칙에 따라 노드와 대화해요. 요청을 보내면 응답을 받습니다. 그런 규칙 가운데 하나가 JSON-RPC예요. Ethereum 네트워크의 노드에게 “이 주소에 코인이 얼마나 있어?”라고 묻는 질문은 이렇게 생겼어요.

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

여기서 `노드_주소`는 우리가 문을 두드리는 곳이고, `지갑_주소`는 누구의 잔액을 묻는지를 뜻해요. `latest`라는 단어는 “가장 최신 시점 기준”이라는 뜻입니다.

노드는 이렇게 대답해요.

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

글자를 아무렇게나 늘어놓은 것 같지만, 16진수로 적은 숫자예요. 바로 1,000,000,000,000,000,000이죠. 노드는 코인을 wei라는 아주 작은 단위로 셉니다. 1 ETH가 딱 이만큼의 wei예요. 그러니까 이 주소에는 1 ETH가 있어요.

결국 비결은 질문과 대답이 전부예요. 남은 문제는 `노드_주소`를 누구의 것으로 할지입니다.

두 갈래 길: 내 노드냐, 제공업체냐

노드는 직접 돌릴 수도 있고, 누군가 대신 운영해 주는 남의 노드에 접속할 수도 있어요. 이런 서비스를 제공업체라고 합니다. 제공업체는 바로 쓸 수 있는 `노드_주소`를 내줘요. 이 책에서 다루는 RpcNode도 그런 서비스 가운데 하나입니다.

질문	내 노드	제공업체
언제부터 쓸 수 있나	설치와 동기화를 마친 뒤. 네트워크에 따라 몇 시간에서 몇 주	가입하고 몇 분 뒤
무엇에 돈을 내나	서버, 디스크, 트래픽, 전문가의 시간	서비스 요금제
누가 업데이트하고 고치나	나	제공업체
네트워크가 여러 개라면	네트워크마다 노드 하나씩	보통 대시보드 하나로 해결
감수하는 위험	자는 사이에 노드가 뒤처지거나 고장 남	남의 서비스에 의존함

두 길 모두 쓸모가 있어요. 특별한 프라이버시가 필요하거나, 부하가 아주 크거나, 네트워크 운영에 직접 참여한다면 내 노드를 두는 게 맞습니다.

> 아르티옴: "저는 커피 볶는 사람이지, 서버 당직 서는 사람이 아니에요. 그래도 직접 하면 대체 얼마가 들까요?"

짧게 답하면, 여러 네트워크를 예비 서버와 전체 기록까지 갖춰 돌리면 수만 달러 단위가 돼요. 작은 노드 하나라면 훨씬 싸게 돌릴 수 있고요. 자세한 계산은 2장에 있어요.

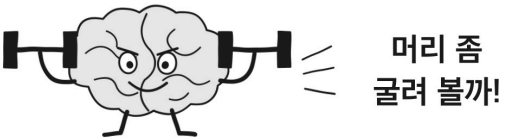
함정: "제공업체에게 지갑 키를 주자"

절대 안 됩니다. 잔액을 확인하거나 결제를 확인하는 데는 주소만 있으면 충분해요. 개인 키와 시드 문구는 돈을 마음대로 움직이는 열쇠라서, 아는 사람이라면 누구나 전부 가져갈 수 있어요. 정직한 제공업체는 이런 것이 필요 없습니다. 키 이야기는 6장에서 할게요.



주의할 점이 하나 더 있어요. 위의 요청은 네트워크의 기본 코인, 여기서는 ETH만 보여 줍니다. 손님이 USDT 같은 토큰으로 결제하면 이 질문으로는 보이지 않아요. 토큰은 다른 방식으로 물어야 하는데, 그 방법은 9장에 있습니다.

생각해 보기



1. 아르티옴이 결제를 확인하는데, 그의 노드가 네트워크보다 한 시간 뒤처져 있어요. 아르티옴은 무엇을 보게 되고, 손님에게 뭐라고 쓰게 될까요?
2. 가게 주문이 하루 세 건이에요. 만약 삼천 건이라면? 내 노드와 제공업체 사이의 선택에서 무엇이 달라질까요?

첫 번째 힌트: 뒤처진 노드는 한 시간 전의 상황을 있는 그대로 보여 줘요.

핵심 정리

- 프로그램은 노드에게 질문해서 블록체인의 일을 알아내요.
- 노드는 직접 둘 수도, 제공업체의 것을 쓸 수도 있어요.
- 노드에게 필요한 것은 주소이지, 개인 키가 아니에요.

다음: 2장. 아르티옴이 계산기를 들고 자체 서버에 얼마가 드는지 따져 봅니다.

제2장. 전부 직접 하면 얼마나 들까?

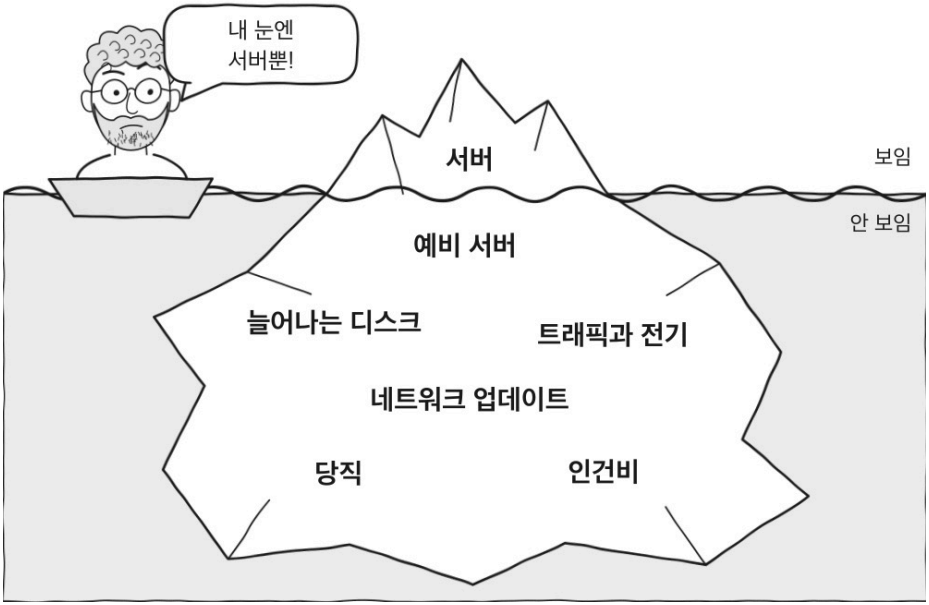
아르티옴, 계산기를 꺼내다

아르티옴의 생각은 단순해요. “서버 사서 노드를 깔면 결제는 알아서 돌아가겠지. 돈은 한 번만 내면 되고.” 그는 계산기를 켜고 하나씩 적기 시작합니다.



알아보기: 비용은 무엇으로 이루어질까

서버는 빙산의 일각일 뿐이에요. 나머지는 처음엔 보이지 않습니다.



1. 서버. 프로세서, 메모리, 빠른 NVMe 디스크.
2. 예비 서버. 한 대가 고장 나면, 고치는 동안 다른 한 대가 대답해요.
3. 디스크는 계속 늘어요. 블록체인에는 새 기록이 쌓이기만 합니다. Bitcoin 체인은 2026년 11월까지 1년 동안 약 700GB에서 786GB로 커졌어요.
4. 트래픽. 노드는 끊임없이 데이터를 주고받아요. TON 라이트서버는 가장 바쁠 때 한 달에 약 16TB가 필요합니다.
5. 업데이트. 네트워크의 규칙이 바뀌면 예전 프로그램은 따라가지 못해요.
6. 사람. 누군가는 이 모든 것을 설정하고, 지켜보고, 밤중에 일어나야 합니다.

네트워크마다 필요한 장비

아래는 공식 문서에 나온 일반 노드의 요구 사양이에요. 네트워크의 투표에 참여하는 노드 기준이 아닙니다.

네트워크	프로세서	메모리	디스크
Bitcoin	1GHz 이상	2GB	750GB 권장. 그런데 체인 자체가 이미 약 786GB
Ethereum	4코어 이상	32GB	NVMe 4TB
BNB Smart Chain	16코어	64GB	SSD 3TB 이상. 전체 DB 스냅샷은 이미 약 7.4TB, 잘라낸 것은 약 1.9TB
TRON	16코어	32GB	SSD 3.5–4TB 이상
Solana (RPC 노드)	16코어 이상	512GB 이상	NVMe 여러 개: 계정 1TB+, 원장 1TB+, 스냅샷 500GB+

이상한 점이 눈에 띄나요? Bitcoin은 체인보다 작은 디스크를 권하고 있어요. BNB는 최소가 3TB인데 전체 스냅샷은 그 두 배가 훌쩍 넘고요. 문서가 늘 네트워크의 속도를 따라가지는 못하기 때문에, 디스크는 넉넉하게 잡습니다.

전체 기록이 필요하다면?

일반 노드는 현재를 잘 알아요. "1년 전 잔액은 얼마였지?"라고 물으려면 아카이브 노드가 필요한데, 이건 10장에서 다룹니다. 아카이브 노드는 훨씬 무거워요. TON 아카이브에는 16–20TB가 필요합니다. Ethereum은 프로그램에 따라 크기 차이가 커서, 2TB에서 14TB 이상까지 돼요. 우리 계산에는 아카이브를 넣지 않았어요. 넣으면 더 비싸집니다.

돈을 세어 봅시다

서버를 살 수도 있지만, 여기서는 임대로 계산할게요. 2026년 11월 27일 기준으로 호스팅 업체들의 공개 가격표를 보고 어림한 값이며, 세금과 설치비는 뺐어요. 가격은 바뀝니다.

서버 등급	구성	월 대략
중형	8코어, 64GB, 2×2TB NVMe	$180
대형	16코어, 128GB, 4–8TB NVMe	$460–560
초대형	메모리 512GB 이상	$1,160–2,350

이제 네트워크 다섯 개를 놓고 봅시다. 가격은 낮은 쪽에 가깝게 잡고, 네트워크마다 서버 두 대(주 서버와 예비 서버)를 둡니다.

네트워크	등급	서버 1대, 월	예비 포함, 월
Bitcoin	중형	$180	$360
Ethereum	중형	$180	$360
BNB Smart Chain	대형	$500	$1,000
TRON	대형	$500	$1,000
Solana	초대형	$1,200	$2,400
합계		$2,560	$5,120

> 아르티옴: “한 달에 5,120달러면 1년에 61,440달러잖아요. 게다가 이게 임대료만이라고요?!”

맞아요. 첫 번째 표에서 높은 쪽 가격을 잡으면 1년에 약 $92,000가 나옵니다. “수만 달러”라는 말은 여기서 나온 거예요.

네트워크 하나만 운영하면 훨씬 쌉니다. Bitcoin은 까다롭지 않아서, 중형보다 소박한 서버로도 충분해요.

그럼 이 일은 누가 하죠?

서버는 스스로를 돌보지 못해요. ‘Habr Career’ 자료에 따르면 러시아에서 중급 DevOps 엔지니어는 한 달에 약 16만~29만 루블을 받습니다. 한 사람이 휴일도 휴가도 없이 24시간 당직을 설 수는 없어요. 두 명만 두어도 1년에 약 380만~700만 루블이 듭니다.

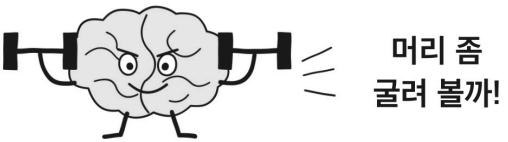
함정: “서버 한 대만 사면 끝”

서버 한 대는 곧 단일 장애점이에요. 디스크가 꽉 차거나, 프로그램이 멈추거나, 네트워크가 업데이트되면 사이트는 결제를 보지 못합니다. 손님은 돈을 보냈는데 주문은 “미결제” 상태로 남는 거죠.



자주 잊는 것이 하나 더 있어요. 결제하기 전에 호스팅 업체의 약관을 읽어 보세요. 어떤 곳은 암호화폐 관련 프로그램을 금지하고, 서버를 꺼 버릴 수도 있습니다.

생각해 보기



1. 아르티옴에게는 Bitcoin만 필요하고, 중형 서버에 예비 서버도 둡니다. 1년에 얼마일까요?
2. 다섯 네트워크에 여섯 번째를 추가했어요. $500짜리 대형 서버를 예비까지 둡니다. 연간 비용은 얼마나 늘까요?
3. 손님이 하루에 세 번 결제해요. 이만한 비용을 들일 가치가 있을까요?

정답: 1) 360 × 12 = $4,320. 2) 1,000 × 12 = $12,000. 3) 결정은 여러분 몫이고, 3장이 도와줄 거예요.

핵심 정리

- 돈은 한 번이 아니라 매달 내야 해요.
- 예비 서버를 갖춘 네트워크 다섯 개는 아카이브와 인건비를 빼고도 임대료만 1년에 $60,000가 넘어요.
- 디스크는 늘고, 네트워크는 업데이트되고, 누군가는 당직을 서야 해요.

다음: 3장. 제공업체가 대신 맡아 주는 일은 무엇이고, 내 노드는 언제 정말 필요할까요.

숫자의 출처

모든 페이지는 2026년 11월 27일에 확인했습니다.

- Bitcoin 요구 사양: bitcoin.org/en/bitcoin-core/features/requirements
- Bitcoin 체인 크기: 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
- 급여: career.habr.com/professions/devops (2025–2026년 자료)

서버 가격은 지은이가 여러 호스팅 업체의 공개 가격표를 보고 반올림해 어림한 값입니다.

제3장. 제공업체는 무엇을 해 주고, 내 노드는 언제 필요할까

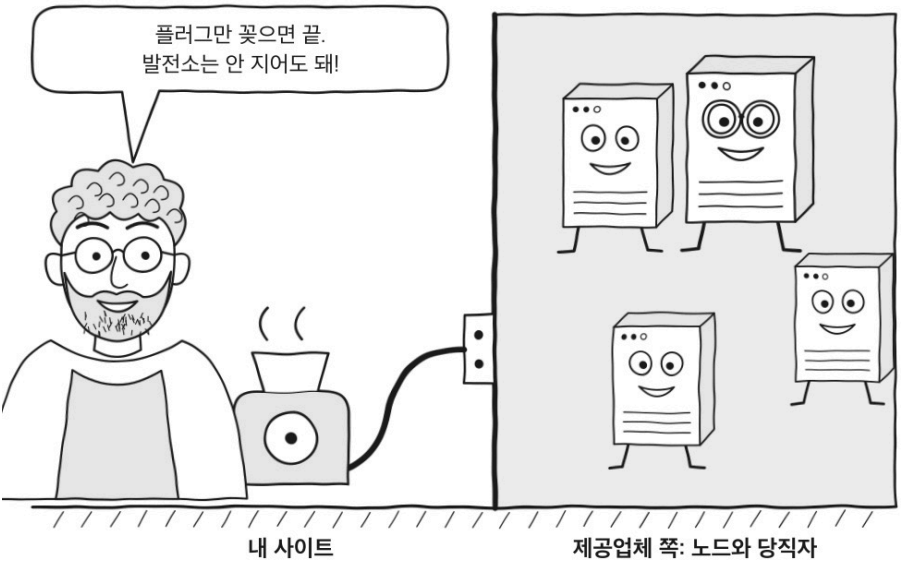
갈림길에 선 아르티옴

2장의 계산을 마친 뒤, 계산기는 화면을 바닥으로 한 채 엎어져 있어요. 예비 서버까지 갖춘 네트워크 다섯 개는 임대료만 1년에 60,000달러가 넘어요. 커피로는 그만큼 벌지 못합니다.



알아보기: 제공업체가 맡아 주는 일

전기는 콘센트에서 받아 쓰지, 주전자 하나 때문에 발전소를 짓지는 않죠. 노드도 마찬가지예요. 제공업체가 노드를 직접 운영하고, 여러분에게는 질문을 보낼 주소를 내줍니다.



보통 제공업체 쪽에서 맡는 일은 다섯 가지예요.

1. 장비와 디스크. 사거나 빌리고, 공간이 모자라면 늘려요.
2. 동기화. 노드가 네트워크에 뒤처지지 않게 해요.
3. 업데이트. 네트워크의 규칙이 바뀌면 제때 프로그램을 바꿔요.
4. 고장 난 것 교체. 요청을 멀쩡한 노드로 보내요.
5. 당직. 밤중에 일어나는 사람은 아르티옴이 아니에요.

제공업체마다 다르고, 약속하는 것도 달라요. 이 목록 가운데 정확히 무엇을 받게 되는지는 돈을 내기 전에 확인하세요. 고르는 방법은 13장에서 살펴봅니다.

여전히 내 몫인 것

- 지갑의 키. 오직 여러분만 가지고 있어요.
- 엔드포인트 주소. 제공업체가 내준 주소를 이렇게 불러요. 보통 이 안에 여러분만의 접근 키가 들어 있습니다. 비밀번호처럼 간수하세요. 남의 요청이 여러분의 한도를 써 버리니까요.
- 내 코드. 오류가 나도 버티고, 요청을 다시 보내고, 결제 한 건을 두 번 인정하지 않아야 해요.

어느 요금제에나 나오는 세 가지 말

용어	뜻	스스로에게 물을 것
초당 요청 수 (RPS)	얼마나 자주 물을 수 있는지	손님이 한꺼번에 몇 명이나 올까?
한도 또는 크레딧	일정 기간에 모두 얼마나 물을 수 있는지. 질문마다 값이 달라요	월말까지 버틸까?
아카이브 접근	먼 과거에 대해 물을 수 있는지	나에게 옛날 기록이 필요한가?

직접 해보기: 진짜 노드에게 질문하기

1장에서는 주소 자리에 `노드_주소` 라고 적었어요. 이제 진짜 주소를 넣어 봅시다. RpcNode에는 가입도 키도 필요 없는 무료 공개 노드가 있어요. Ethereum 네트워크의 마지막 블록 번호를 물어볼게요.

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

이 장을 쓰던 시점에 노드는 이렇게 대답했어요.

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

`0x1946393` 은 26,502,035예요. 여러분이 받는 숫자는 더 클 거예요. 네트워크는 가만히 있지 않으니까요. 1분 뒤에 다시 요청해 보면 직접 확인할 수 있어요.

공개 노드는 모두가 함께 쓰기 때문에 제한이 있어요. 2026년 11월 27일 기준으로 주소 하나당 초당 5건이고, 가장 "비싼" 메서드는 막혀 있습니다. 맛보기로는 충분해요. 가게를 하려면 나만의 엔드포인트가 필요합니다.

비용은 얼마일까

예로 2026년 11월 27일 기준 RpcNode의 처음 두 요금제를 볼게요. 최신 요금은 사이트에서 확인하세요.

	Free	Starter
월 요금	$0	$19
크레딧	1,500만	5,000만
초당 요청 수	15	40
엔드포인트	3	5
네트워크	Ethereum과 Solana	전체
아카이브 접근	없음	있음

RpcNode의 노드는 세 지역에 있어요. 프랑크푸르트, 미국, 일본입니다.

> 아르티옴: “한 달에 5,120달러 대신 19달러라고요? 무슨 꿍꿍이가 있죠?”

서로 다른 물건이에요. 내 서버는 통째로 내 것이지만, 요금제에서는 한도가 정해진 공용 자원의 한 몫을 받는 거예요. 작은 가게에는 그 몫이면 충분하고, ‘발전소’ 하나 값을 통째로 낼 이유가 없습니다.

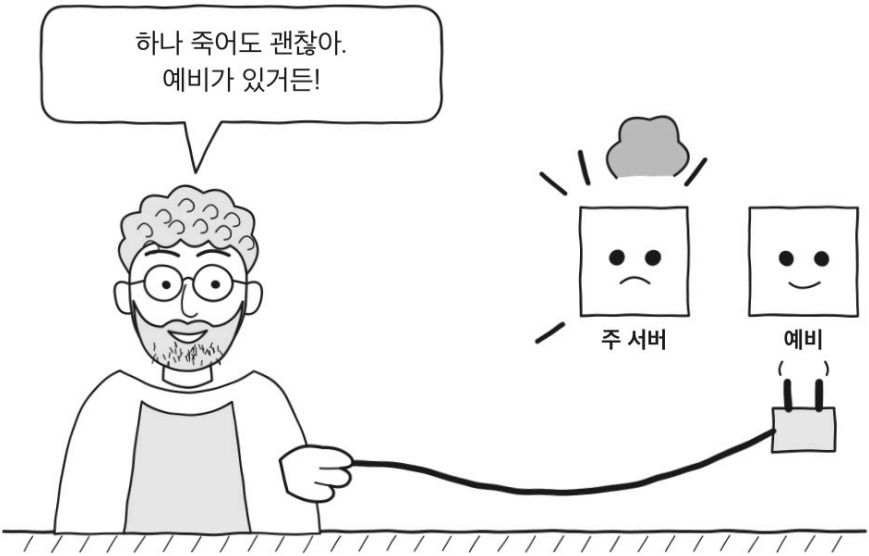
그래도 내 노드가 필요한 경우

- 프라이버시. 제공업체는 여러분의 요청을 볼 수 있어요. 어떤 주소를 확인하는지, 어디에서 묻는지요.
- 어마어마한 부하. 요청이 너무 많아서 요금제가 자체 서버보다 비싸져요.
- 네트워크 참여. 검증자이거나 네트워크 자체를 개발하는 경우예요.
- 특별한 설정. 제공업체가 주지 않는 메서드나 모드가 필요해요.
- 법이나 고객사의 요구. 데이터를 반드시 내 쪽에서만 처리해야 해요.

섞어 쓰는 길도 있어요. 가장 중요한 네트워크는 내 노드로, 나머지와 만일의 대비는 제공업체로 하는 거죠.

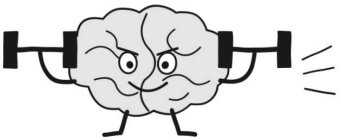
함정: “제공업체는 절대 안 죽어”

무엇이든 죽을 수 있어요. 서버도, 통신망도, 제공업체의 서비스도요. 결제가 중요하다면, 대답이 없을 때 사이트가 무엇을 할지 미리 정해 두세요.



- 그저 물어보지 못했을 뿐인데 손님에게 “결제되지 않았습니다”라고 말하지 마세요.
- 요청은 잠시 뒤에 다시 보내세요. 연달아 백 번씩 보내지 말고요.
- 예비 주소를 준비하세요. 두 번째 제공업체나 내 노드요.

생각해 보기



머리 좀 굴려 볼까!

1. 아르티옴의 사이트는 결제를 기다리는 주문 20건을 각각 1초에 한 번씩 확인하고, 확인 한 번에 요청 하나를 씁니다. 초당 15건 안에 들어갈까요?
2. 노드가 10분 동안 대답하지 않으면, 손님에게는 무엇이 보여야 할까요?
3. 커피 가게와 네트워크 자체를 개발하는 팀, 둘 중 누구에게 자기 노드가 필요할까요?

첫 번째 힌트: 20은 15보다 커요. 덜 자주 확인하거나 더 똑똑하게 물어야 하는데, 방법은 26장에 있어요.

핵심 정리

- 제공업체는 장비, 동기화, 업데이트, 당직을 맡아요.
- 키, 엔드포인트 주소, 장애가 났을 때 사이트의 행동은 여전히 내 몫이에요.
- 내 노드는 프라이버시, 큰 부하, 네트워크 참여, 특별한 요구 때문에 필요해요.

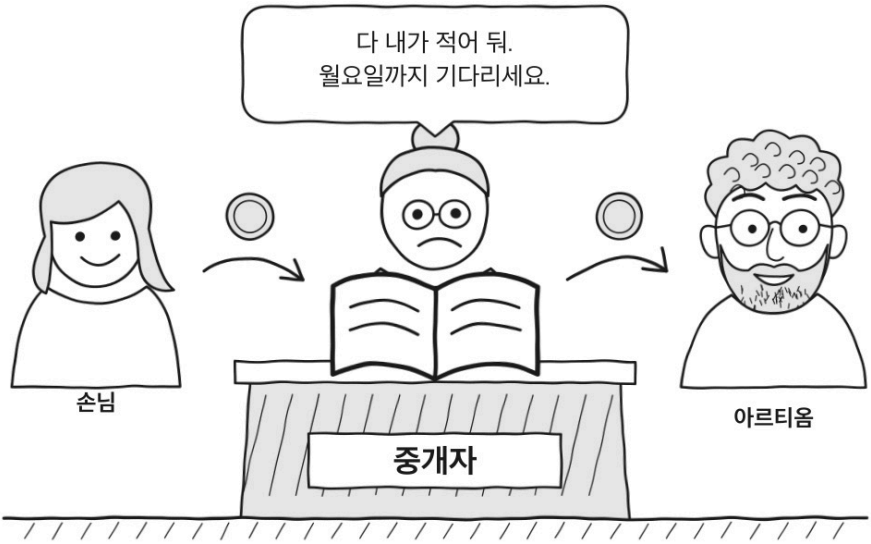
파트 1 끝. 아르티옴은 제공업체를 골랐고, 진짜 노드에게서 대답을 받았어요. 그런데 노드가 말한 26,502,035번 '블록'이란 대체 뭘까요? 다음: 파트 2, 4장. 중개자 없는 돈과 기록.

제4장. 중개자 없는 돈과 기록

여러분의 돈은 어디에 있을까?

아르티옴이 카드로 입금을 받았어요. 지폐는 한 장도 어디로 움직이지 않았죠. 그럼 무슨 일이 일어난 걸까요? 누군가 손님 이름 옆의 숫자를 줄이고, 아르티옴 이름 옆의 숫자를 늘린 거예요.

현금이 아닌 돈은 곧 기록이에요. 그리고 그 기록을 관리하는 쪽을 중개자라고 합니다. 은행이나 결제 서비스 같은 곳이죠.



알아보기: 중개자는 왜 필요하고, 무엇이 불편할까

중개자는 쓸모 있는 일을 많이 해요. 돈이 충분한지 확인하고, 같은 돈을 두 번 쓰지 못하게 막아요. 분쟁을 조정해 결제를 되돌려 주기도 하죠.

하지만 모든 기록이 중개자의 손에 있어요. 송금을 멈출 수도, 수수료를 뗄 수도, 계좌를 막을 수도 있고, 휴일에는 문을 닫기도 해요. 손님이 다른 나라에 살면 이런 점이 특히 두드러집니다.

> 아르티옴: “그럼 손님이 돈을 파일로 보내 주면 되잖아요. 사진처럼요. 중개자 같은 거 없이!”

왜 파일을 그냥 보내면 안 될까

사진은 백 번을 보내도 내게 그대로 남아요. 돈이 그러면 큰일이죠. 동전 하나로 두 사람에게 값을 치를 수 있게 되니까요. 이것을 이중 지불이라고 합니다.

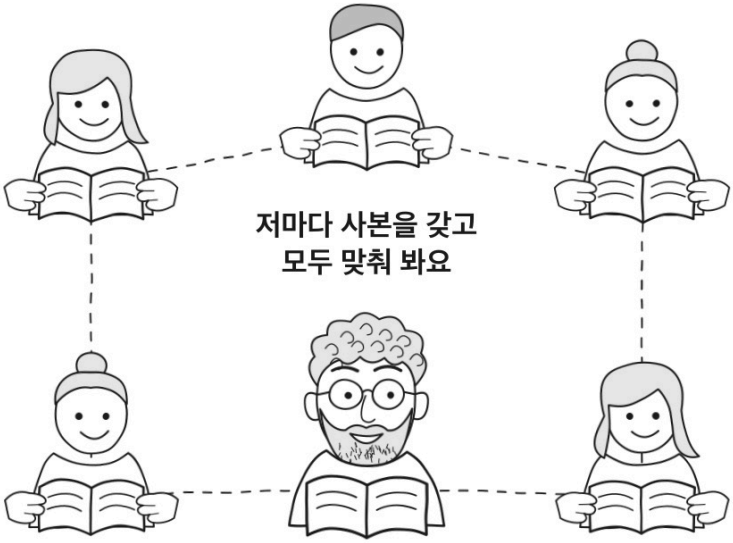


오랫동안 해결책은 하나뿐이었어요. 대장 하나가 목록을 관리하면서 지금 그 동전이 누구 것인지 말해 주는 거죠. 결국 다시 중개자예요.

발상: 모두가 기록을 보관한다

2008년, 사토시 나카모토라는 가명의 저자가 다른 길을 설명했고, 2009년 1월에 Bitcoin 네트워크가 돌아가기 시작했어요. 발상은 이렇습니다.

1. 송금 목록을 한 사람이 감춰 두지 않아요. 많은 사람이 사본을 가집니다.
2. 새 송금이 생길 때마다 참가자들이 공통 규칙에 따라 검사해요. 돈이 있는지, 이미 쓴 돈은 아닌지요.
3. 대장 없이도, 어떤 기록이 맞는지 참가자들이 합의합니다.



이렇게 여럿이 함께 관리하는 공용 목록을 블록체인이라고 불러요. 기록이 어떻게 사슬로 이어지는지는 5장에서, 참가자들이 어떻게 합의하는지는 8장에서 살펴봅니다.

직접 해보기: 공용 장부 놀이

코드는 필요 없어요. 친구 두 명을 불러서 각자 종이를 한 장씩 갖게 하세요. 처음에는 모두 이렇게 적어 둡니다. 아냐는 10코인, 보리스는 5코인, 베라는 0코인. 규칙은 하나예요. 가진 것보다 많이 줄 수 없다.

번호	누가 보내려 하나	받아들일까?	아냐	보리스	베라
시작			10	5	0
1	아냐 → 베라 4				
2	베라 → 보리스 1				
3	보리스 → 아냐 7				
4	아냐 → 보리스 6				

빈칸을 채운 뒤 서로의 종이를 비교해 보세요. 모두 똑같다면, 여러분은 방금 대장 없이 합의를 이룬 거예요. 정답은 이 장 끝에 있어요.

블록체인이 주는 것과 주지 않는 것

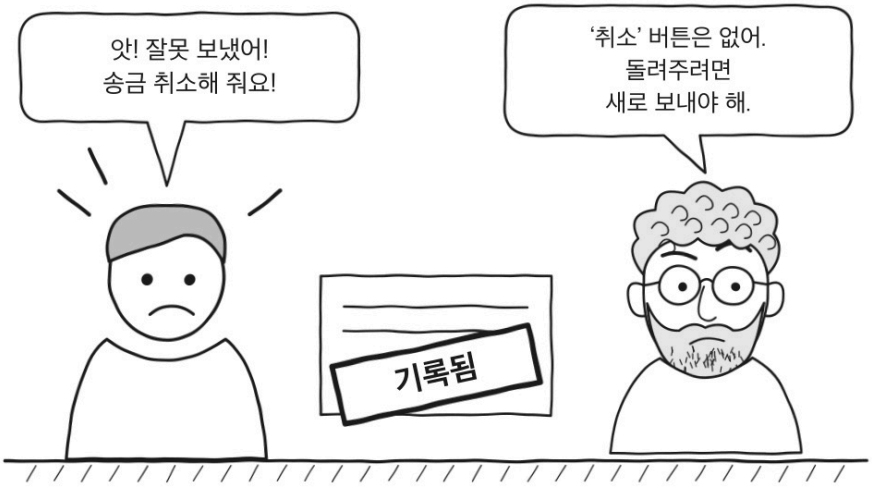
주는 것	주지 않는 것
어느 날 어느 시각이든 직접 송금	'취소' 버튼
여러 나라에서 접근할 수 있는 하나의 계정	고정된 코인 가격: 시세는 크게 출렁일 수 있어요
누구나 확인할 수 있는 기록	비밀: 어떤 주소의 송금이든 모두에게 보여요
목록 보관을 누구에게 맡겨 믿을 필요가 없음	잃어버린 키를 되찾아 주는 고객센터

모두가 출렁이는 것은 아니에요. 1달러 언저리를 유지하려는 코인도 있는데, 이를 스테이블코인이라고 합니다. 하지만 가장 널리 쓰이는 것들 뒤에는 발행 회사가 있고, 그 회사는 주소에 있는 돈을 동결할 수 있어요. 자세한 내용은 9장에 있습니다.

중개자가 사라진 것은 아니에요. 거래소, 지갑, 제공업체가 있죠. 하지만 서로 달라요. 거래소는 여러분의 코인을 직접 보관할 수 있어요. 파트 1의 제공업체는 돈을 보관하지 않고, 여러분의 질문을 노드에 전달할 뿐입니다.

함정: "마음에 안 들면 취소하지 뭐"

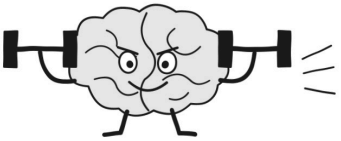
컨펌된 송금은 취소할 수 없어요. 주소를 잘못 적으면 돈은 가 버립니다. 돌려줄 수 있는 사람은 받은 사람뿐이에요. 그것도 그가 원해야 하고, 애초에 주인이 있는 주소여야 합니다.



그래서 주소는 두 번 확인하고, 큰 금액은 먼저 조금만 보내 봅니다.

짧은 이야기 하나. 2010년 5월, 누군가 피자 두 판 값으로 10,000비트코인을 냈어요. 그 송금은 지금도 공용 장부에 적혀 있습니다.

생각해 보기



머리 좀 굴려 볼까!

1. 놀이가 끝난 뒤 베라에게는 3코인이 있어요. 베라가 동시에 아냐에게 “3코인 줄게”, 보리스에게 “3코인 줄게”라고 씁니다. 참가자들은 어떻게 해야 할까요?
2. 어떤 참가자는 첫 번째 송금을, 다른 참가자는 두 번째를 받아들였다면? 누가 맞을까요?
3. 아르티옴에게는 결제를 취소할 수 있는 게 편할까요, 없는 게 편할까요? 손님에게는요?

힌트: 1) 하나만 받아들일 수 있어요. 2) 가장 어려운 질문이에요. 8장 전체가 이 이야기입니다.

핵심 정리

- 현금이 아닌 돈은 기록이고, 누군가는 그 기록을 관리해야 해요.
- 블록체인은 여럿이 공통 규칙에 따라 함께 관리하는 공용 목록이에요.
- 중개자에게서 자유롭다는 것은 책임도 진다는 뜻이에요. 송금을 취소해 주거나 키를 되찾아 줄 사람이 없어요.

놀이 정답: 1번 송금 승인 (6, 5, 4), 2번 승인 (6, 6, 3), 3번 거절. 보리스에게는 6뿐이에요 (6, 6, 3). 4번 승인 (0, 12, 3). 코인 총합은 여전히 15예요.

다음: 5장. 노드가 말한 ‘블록’이란 무엇이고, 왜 옛 기록을 몰래 고칠 수 없을까요.

제5장. 블록과 체인: 옛 기록을 몰래 고칠 수 없는 이유

바로 그 번호

3장에서 노드는 아르티옴에게 마지막 블록이 26,502,035번이라고 대답했어요. 블록이 뭘까요? 왜 번호가 붙어 있을까요? 그리고 누군가 장부의 기록을 몰래 고치지 못하게 막는 것은 무엇일까요?

알아보기: 블록은 장부의 한 페이지

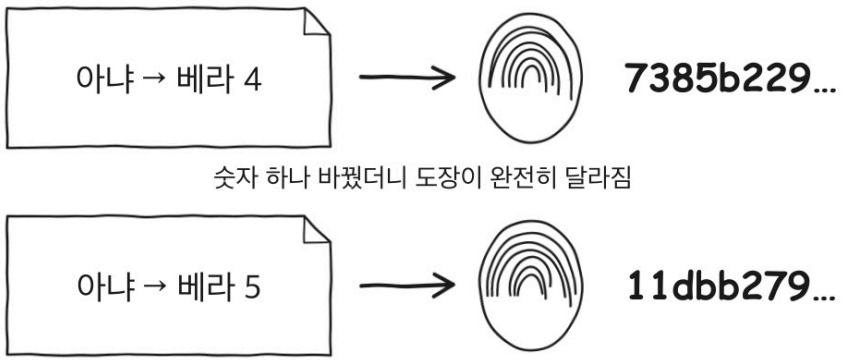
송금은 하나씩 따로 적지 않아요. 한 페이지에 여러 줄을 적듯이 묶음으로 모읍니다. 이런 페이지를 블록이라고 해요. 블록에는 이런 것이 있습니다.

1. 송금과 그 밖의 작업. 이것들을 트랜잭션이라고 불러요.
2. 순서대로 붙인 페이지 번호.
3. 만들어진 시각.
4. 이전 페이지의 도장. 모든 비밀이 여기에 있어요.

도장, 곧 해시

어떤 글이든 숫자와 글자로 된 짧은 문자열을 계산해 낼 수 있어요. 이것이 해시인데, 여기서는 도장이라고 부를게요. 지문과 비슷합니다.

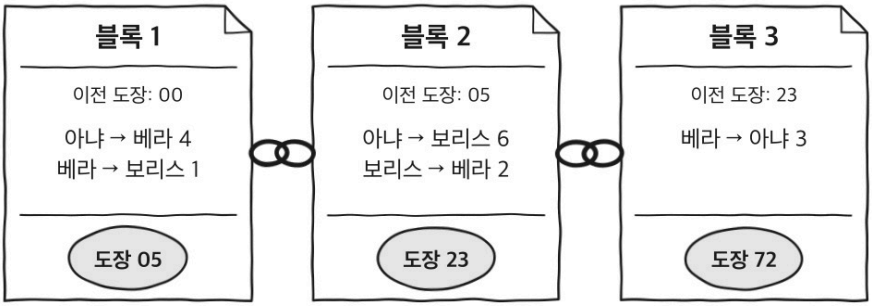
- 같은 글이면 도장은 언제나 같아요.
- 글자 하나만 바꿔도 도장이 완전히 달라져요.
- 도장만 가지고 원래 글을 되살릴 수는 없어요.
- 같은 도장이 나오는 다른 글을 찾아내는 일은 현실적으로 불가능해요.



그림에 있는 두 문자열의 실제 SHA-256 해시 앞부분이에요. SHA-256은 Bitcoin이 쓰는 방식이고, 다른 네트워크는 다른 방식으로 도장을 계산하기도 합니다.

체인

각 페이지에는 앞 페이지의 도장이 적혀 있어요. 페이지 자신의 도장은 거기 담긴 모든 것으로 만드는데, 이 앞 페이지의 도장까지 포함해서 만듭니다.

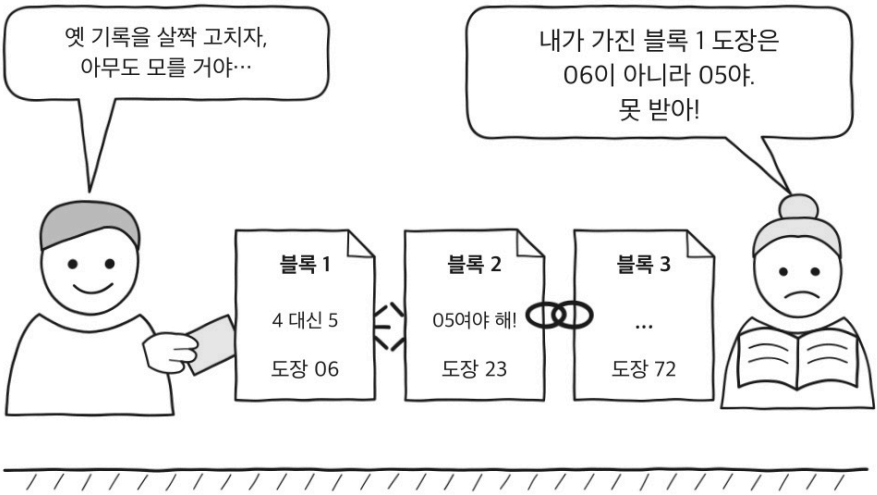


이렇게 페이지들이 서로 맞물려 블록의 체인이 돼요. 영어로 blockchain. 이름의 유래는 이게 전부입니다.

그림 속 도장은 두 자리 숫자로 된 장난감 도장이에요. 암산할 수 있도록요.

옛 기록을 살짝 고치면 어떻게 될까

사기꾼이 첫 번째 블록의 4를 5로 바꿔요. 블록의 도장이 바뀝니다. 그런데 두 번째 블록에는 옛 도장이 적혀 있어요. 체인이 끊어지고, 바로 눈에 띕니다.



사기꾼이 뒤따르는 도장을 모두 다시 쓸 수도 있어요. 하지만 다른 참가자들에게는 각자의 사본이 있고, 사기꾼의 버전은 그것들과 맞지 않아요. 자기 버전을 받아들이게 하려면 나머지 네트워크 전체를 힘으로 이겨야 합니다. 비용이 엄청나죠. 네트워크가 어느 체인이 맞는지 정확히 어떻게 결정하는지는 8장에서 살펴봅니다.

진짜 체인을 들여다보기

2026년 11월 26일 Ethereum 네트워크의 블록 세 개예요. 도장이 길어서 앞과 끝만 보여 드립니다.

블록 번호	자기 도장	이전 블록의 도장	트랜잭션 수
26,502,033	`0xbe58…d764`	`0x2844…c876`	229
26,502,034	`0x7bb7…b7a4`	`0xbe58…d764`	229
26,502,035	`0xeab2…c904`	`0x7bb7…b4a4`	515

손가락으로 따라가 보세요. 한 줄의 도장이 다음 줄에 다시 나옵니다. 이 블록들 사이에는 정확히 12초씩 흘렀어요. Bitcoin에서는 새 블록이 더 드물게, 평균 약 10분에 하나씩 나옵니다.

맨 첫 블록에는 이전 블록이 없어요. Bitcoin의 첫 블록은 2009년 1월 3일에 만들어졌습니다.

직접 해보기: 나만의 도장

장난감 규칙을 하나 만들어 볼게요.

> 도장 = (이전 도장 × 3 + 페이지의 모든 금액)의 결과에서 끝 두 자리.

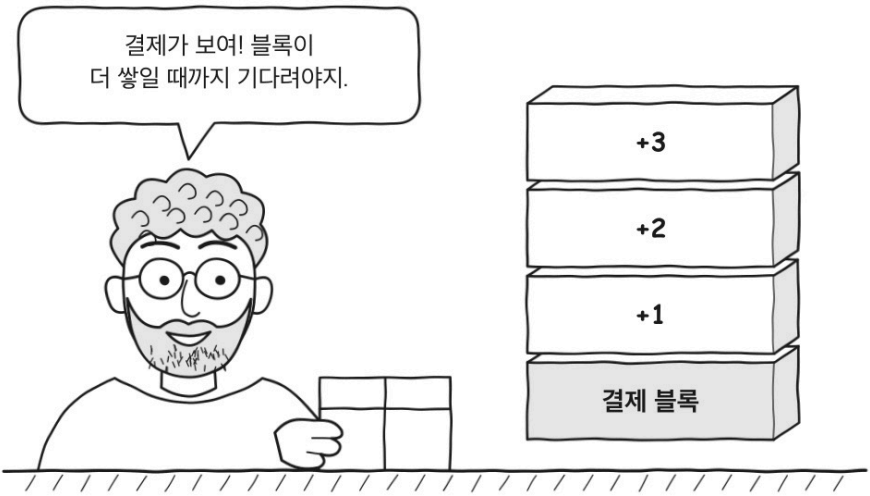
블록	송금 금액	도장	블록 1의 4를 5로 바꾸면
1	4와 1		
2	6과 2		
3	3		

첫 번째 블록의 이전 도장은 0이에요. 두 열을 모두 채워 보세요. 도장이 몇 개나 바뀌었나요?

함정: "블록에 보였으니 영원하겠지"

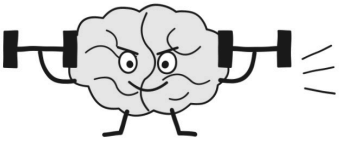
가장 최근의 블록들은 가끔 다른 블록으로 바뀌어요. 네트워크의 일부가 아직 합의를 마치지 못한 거예요. 위에 블록이 많이 쌓일수록 내 기록을 없애기가 어려워집니다.

이것을 컨펌으로 셉니다. 결제가 담긴 블록 자체가 첫 번째 컨펌이고, 다음 블록이 하나씩 붙을 때마다 컨펌이 하나씩 늘어요.



얼마나 기다릴지는 네트워크와 금액에 따라 달라요. 커피 한 잔이면 적게, 큰 도매 주문이면 더 오래요. 자세한 내용은 8장에 있습니다.

생각해 보기



머리 좀 굴려 볼까!

1. 우리 도장을 속여 보세요. 블록 1의 금액 4와 1을 도장이 바뀌지 않는 다른 값으로 바꿔 보세요. 이것은 우리 규칙에 대해 무엇을 말해 줄까요?
2. 3년 전 블록을 바꾸는 것이 왜 어제 블록보다 어려울까요?
3. 블록이 12초마다 하나씩 나온다면, 한 시간에 블록이 몇 개 생길까요?

정답: 1) 예를 들어 3과 2. 합이 같아요. 진짜 해시라면 알아챘을 거예요. 2) 위에 쌓인 것을 전부 다시 만들어야 하니까요. 3) 300개.

핵심 정리

- 블록은 트랜잭션, 번호, 시각, 이전 페이지의 도장이 담긴 페이지예요.
- 옛 기록을 바꾸면 도장이 바뀌고 체인이 끊어져요.
- 갓 만들어진 블록은 덜 믿고, 컨펌을 기다려요.

놀이 정답: 도장은 05, 23, 72. 바꾼 뒤에는 06, 26, 81로 세 개 모두 바뀌었어요.

다음: 6장. 송금을 보낸 사람이 정말 돈의 주인인지 네트워크는 어떻게 알까요? 키, 주소, 서명 이야기입니다.

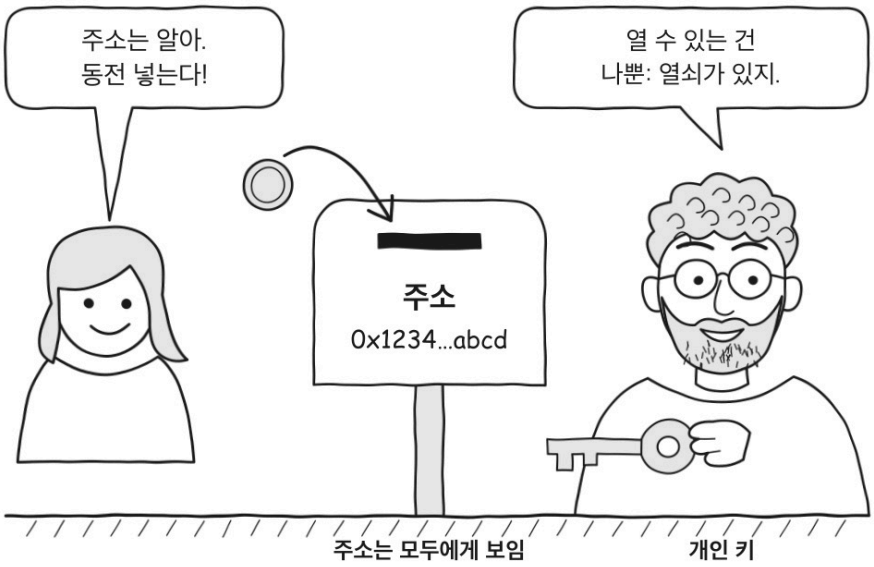
제6장. 키, 주소, 서명

네트워크는 내 돈인 줄 어떻게 알까?

공용 장부에는 신분증이 없어요. 아무도 이름을 묻지 않고 사진을 대조하지도 않죠. 그렇다면 누구든 "아르티옴이 나에게 전 재산을 준다"라고 적지 못하게 막는 것은 뭘까요?

알아보기: 우편함과 열쇠

길가의 우편함을 떠올려 보세요. 우편함 번호는 누구나 알고, 누구든 투입구에 동전을 넣을 수 있어요. 하지만 우편함을 열고 동전을 꺼낼 수 있는 사람은 열쇠를 가진 사람뿐입니다.



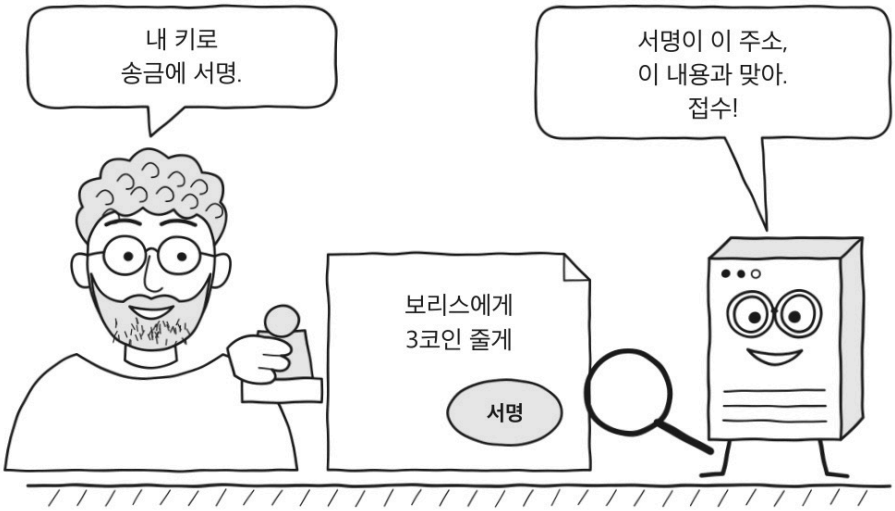
무엇	비유하면	누구에게 보여 주나
주소	우편함 번호	돈을 받으려면 누구에게든
공개 키	서명을 확인하는 견본	모두에게 보여도 됨
개인 키	우편함 열쇠	아무에게도 안 됨

이 셋은 한 방향으로 이어져 있어요. 개인 키에서 공개 키를 만들고, 공개 키에서 주소를 만듭니다. 거꾸로 계산하는 것은 현실적으로 불가능해요. 어떤 네트워크에서는 주소가 곧 공개 키이고, 특별한 방식으로 적었을 뿐이에요.

지갑은 키를 보관하는 프로그램이나 기기예요. 지갑 안에 코인이 들어 있는 것은 아닙니다. 코인은 공용 장부의 기록으로 남아 있어요.

서명

코인을 내주려면 주인이 송금 내용을 작성하고 개인 키로 서명해요. 이때 키 자체는 어디에도 보내지 않습니다. 네트워크로 나가는 것은 송금 내용과 서명뿐이에요.



노드는 두 가지를 확인해요. 서명이 바로 그 주소의 키로 만든 것인지, 그리고 서명한 뒤에 내용이 바뀌지 않았는지요. 한 송금의 서명은 다른 송금에 맞지 않아요. 그리고 똑같은 송금을 네트워크가 두 번 처리하지도 않습니다.

직접 해보기: 장난감 서명

전부 계산기로 할 수 있어요. 개인 키는 숫자 7, 공개 키는 숫자 3입니다. 비밀은 앞의 것뿐이에요.

> 규칙: 정해진 숫자를 곱하고, 단계마다 33으로 나눈 나머지만 남기세요.

메시지 "4"에 서명하기. 4를 개인 키가 시키는 횟수만큼, 즉 7번 가져옵니다.

4의 개수	1	2	3	4	5	6	7
나머지	4	16	31	25	1	4	16

각 단계는 직전 나머지에 4를 곱한 거예요. 예를 들어 세 번째는 16 × 4 = 64이고, 33으로 나눈 나머지는 31입니다. 일곱 번째 단계가 끝나면 서명 16이 나와요.

확인하기. 서명 16을 공개 키가 시키는 횟수, 즉 3번 가져옵니다. 16, 그다음 16 × 16 = 256에서 나머지 25, 그다음 25 × 16 = 400에서 나머지 4. 우리 메시지가 나왔어요. 서명은 진짜입니다!

이번에는 누군가 서명 16은 그대로 두고 메시지를 "5"로 고쳤어요. 확인해 보면 여전히 5가 아니라 4가 나옵니다. 위조가 드러나요.

원리만 보여 주는 예예요. 이렇게 작은 숫자는 학생도 깰 수 있어요. 실제 네트워크의 키는 어마어마하게 큰 숫자이고, 쓰는 수학도 다릅니다.

시드 문구

개인 키는 긴 문자열이라 옮겨 적기가 불편해요. 그래서 많은 지갑은 12개 또는 24개 단어로 된 시드 문구를 줍니다. 이 문구에서 키와 모든 주소가 복구돼요.



- 문구를 아는 사람이 돈의 주인이에요.
- 문구와 기기를 모두 잃어버리면 되살려 줄 사람이 없어요.
- 사진 찍지 말고, 메모 앱이나 메신저에 보관하지 마세요.
- 웹사이트에 입력하지 말고, '고객센터'에 불러 주지 마세요. 진짜 고객센터는 묻지 않아요.

비밀번호는 바꿀 수 있지만 개인 키는 바꿀 수 없어요. 키가 새어 나갔다면, 급히 새 주소로 돈을 옮기는 수밖에 없습니다.

가게에는 무슨 뜻일까

돈을 받고 결제를 확인하는 데 아르티옴의 사이트에 필요한 것은 주소뿐이에요. 가게 서버에 개인 키는 필요 없고, 두어서도 안 됩니다. 사이트가 해킹당하면 돈까지 가져가니까요. 주문마다

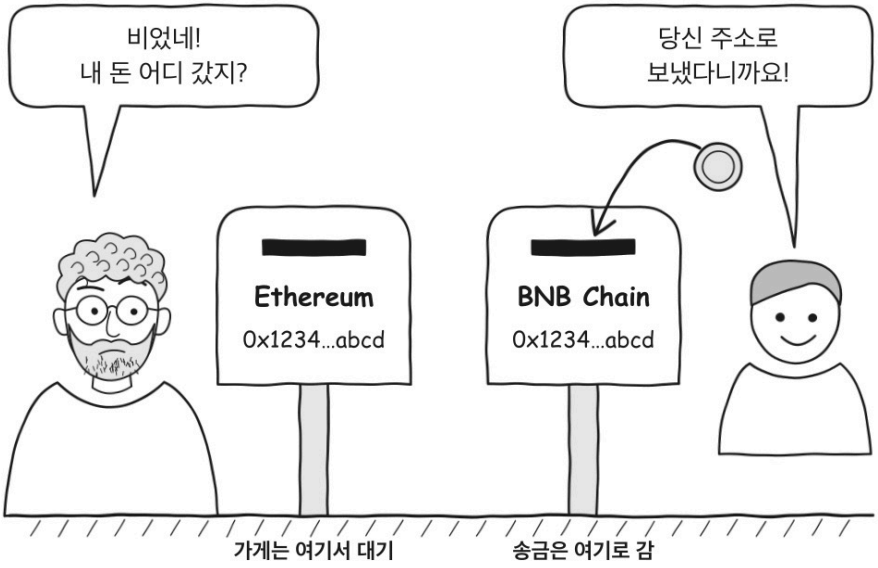
주소를 따로 내주면 편한데, 이건 28장에서 다룹니다.

함정: "주소가 같으니 잘 도착하겠지"

네트워크마다 주소 모양이 달라요.

네트워크	주소의 시작
Bitcoin	`1`, `3` 또는 `bc1`
Ethereum과 비슷한 네트워크	`0x`
TRON	`T`
TON	보통 `EQ` 또는 `UQ`
Solana	특별한 시작 글자가 없음

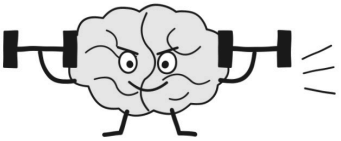
하지만 Ethereum, BNB Smart Chain, Polygon을 비롯한 여러 네트워크에서는 주소가 똑같이 생겼어요. 그런데 장부는 서로 다릅니다. 손님이 다른 네트워크로 보내면 가게는 결제를 보지 못해요.



그래서 청구서에는 주소와 금액뿐 아니라 네트워크도 적어요.

사기꾼의 수법이 하나 더 있어요. 여러분이 아는 주소와 비슷하게 생긴 주소에서 아주 작은 금액을 보냅니다. 거래 내역에서 그 주소를 복사하기를 노리는 거예요. 앞뒤 몇 글자만 보지 말고 주소 전체를 대조하세요.

생각해 보기



머리 좀 굴려 볼까!

1. 메시지 “2”에 키 7로 서명하고, 키 3으로 서명을 확인해 보세요.
2. 아르티옴이 가게 주소를 명함에 인쇄했어요. 위험할까요?
3. ‘지갑 고객센터’가 멈춘 송금을 돌려주겠다며 시드 문구를 달라고 해요. 어떻게 해야 할까요?

정답: 1) 서명은 29이고, 확인하면 다시 2가 나와요. 2) 그것만으로 돈을 훔칠 수는 없지만, 그 주소로 들어오는 돈은 모두가 보게 돼요. 3) 아무것도 알려 주지 마세요. 사기꾼이에요.

핵심 정리

- 주소는 누구에게나 보여 주고, 개인 키와 시드 문구는 아무에게도 보여 주지 않아요.
- 서명은 키의 주인이 송금을 허락했고 내용이 바뀌지 않았음을 증명해요.
- 네트워크는 주소만큼 중요해요.

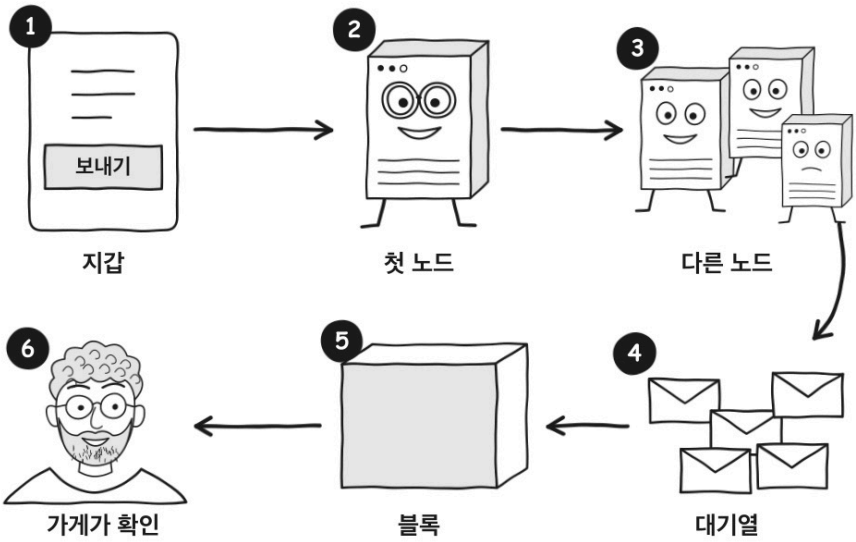
다음: 7장. 송금에 서명했어요. ‘보내기’ 버튼에서 블록에 기록되기까지 무슨 일이 벌어질까요?

제7장. 트랜잭션의 여정: '보내기' 버튼에서 블록까지

"보냈어요!" — "제 쪽엔 아무것도 없는데요"

손님이 버튼을 누르고 "보냈어요"라고 써요. 아르티옴이 확인하니 돈이 없어요. 누가 속이는 걸까요? 아마 아무도 아닐 거예요. 송금이 아직 가는 중입니다.

알아보기: 여섯 개의 정거장



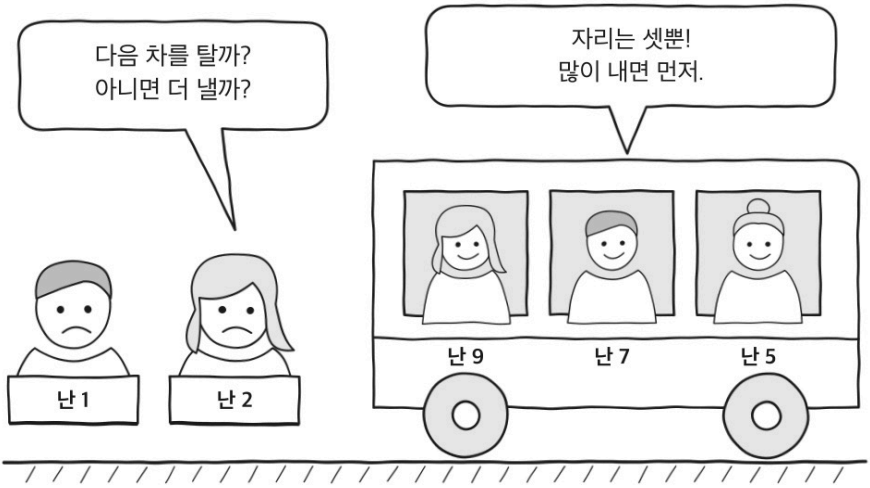
1. 지갑이 송금 내용을 작성하고 키로 서명해요.
2. 첫 번째 노드가 송금을 받아 서명이 맞는지, 돈이 충분한지 검사해요.
3. 그 노드는 다른 노드들에게 전달하고, 그 노드들은 또 다음으로 전달해요.
4. 송금은 대기열에서 기다려요. 영어로는 mempool(멤풀)이라고 합니다.
5. 페이지를 모을 차례가 된 참가자가 송금을 블록에 넣어요.
6. 가게가 송금을 보고 컨펌을 기다려요.

네트워크마다 이름과 세부 사항은 달라요. 예를 들어 어떤 네트워크에는 공용 대기열이 없고, 송금이 곧바로 블록을 모으는 쪽으로 갑니다.

모든 송금에는 자기만의 '도장'이 있어요. 바로 트랜잭션 해시예요. 택배의 운송장 번호 같은 것이라, 이것으로 송금이 지금 어디 있는지 알 수 있습니다.

수수료: 버스 좌석

블록에 모든 것이 다 들어가지는 않아요. 좌석이 세 개인 버스를 떠올려 보세요. 정류장에는 다섯 명이 서 있고요.



블록에 타려면 보내는 사람이 네트워크 수수료를 내요. 원하는 사람이 많을수록 자리가 비싸집니다. 적게 부른 사람은 다음 버스를 기다리고, 가끔은 끝내 못 타기도 해요.

- 수수료는 보내는 사람이 내고, 가게가 받는 것이 아니에요.
- 수수료는 송금 금액이 아니라 송금의 '무게'와 네트워크의 혼잡도에 달려 있어요.
- 이 '무게'를 계산하는 방법은 네트워크마다 달라요. 파트 4에서 다룹니다.

진짜 송금 들여다보기

Ethereum 네트워크의 평범한 송금이에요. 블록은 26,502,085번입니다. 트랜잭션 해시로 노드에게 이렇게 물었어요.

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

응답은 길어요. 핵심만 골라 보통의 숫자로 바꾸면 이렇습니다.

필드	값	뜻
`status`	`0x1`	성공적으로 실행됨
`blockNumber`	26,502,085	어느 블록에 들어 있는지
`gasUsed`	21,000	단순 송금의 '무게'
수수료	0.000043 ETH	'무게' × 단위당 가격
송금 금액	0.000041 ETH	송금 트랜잭션 자체에서 읽음

눈치채셨나요? 이번에는 보낸 돈보다 보내는 데 낸 돈이 더 많아요. 수수료는 안에 돈이 얼마나 들었는지 신경 쓰지 않습니다.

그리고 이 숫자는 늘 바뀌어요. 직접 물어보세요.

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

노드는 '무게' 한 단위의 현재 가격을 wei로 알려 줘요. 같은 날 RpcNode의 Bitcoin 공개 노드에서는 거의 83,000건의 송금이 대기열에서 기다리고 있었습니다.

송금에 일어날 수 있는 일

상태	뜻	가게가 할 일
대기 중	아직 대기열에 있음	기다리고, 상품은 보내지 않기
블록에 포함	기록됐지만 컨펌이 적음	성공 여부를 확인하고 기다리기
컨펌 완료	위에 블록이 충분히 쌓임	주문 발송하기
실행 실패	블록에는 있지만 송금은 이루어지지 않음	결제로 치지 않기
대기열에서 탈락 또는 교체	블록에 들어가지 못함	손님에게 지갑을 확인해 달라고 하기

함정: "블록에 들어갔으니 결제 완료"

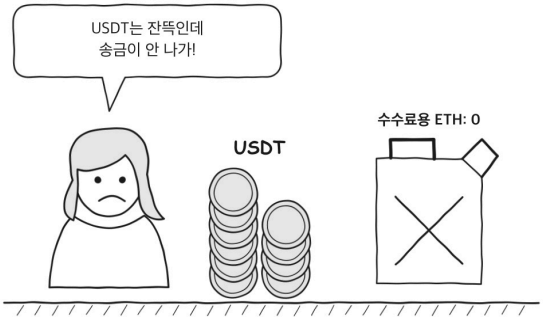
Ethereum과 비슷한 네트워크에서는 송금이 블록에 들어가고도 실행되지 않을 수 있어요. 예를 들어 손님의 토큰이 모자란 경우죠. 이때도 수수료는 빠져나가지만, 돈은 여러분에게 오지 않아요.



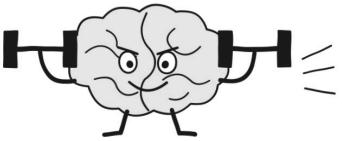
그래서 블록만 보지 말고 결과도 봐야 해요. 위 예제에서는 `status` 필드입니다. 받는 사람, 금액, 코인, 네트워크도 함께 대조하세요.

두 번째 함정: 텅 빈 연료통

수수료는 네트워크의 기본 코인으로 냅니다. Ethereum에서는 USDT를 보내더라도 ETH로 내요. TRON에서는 TRX와 네트워크의 특별한 자원으로 냅니다.



생각해 보기



머리 좀 굴려 볼까!

1. 블록에는 송금 세 건이 들어가요. 대기열의 사람들이 자릿값으로 4, 9, 1, 7, 2를 불렀어요. 첫 블록에는 누가 탈까요?
2. 손님이 트랜잭션 해시와 “성공”이라고 뜬 화면 캡처를 보냈어요. 이것만으로 커피를 보내도 될까요?
3. 손님에게 Ethereum 네트워크의 USDT 100개가 있는데 ETH는 한 방울도 없어요. 어떻게 해야 할까요?

정답: 1) 9, 7, 4. 2) 안 돼요. 화면 캡처는 위조할 수 있으니, 해시를 내 쪽에서 직접 확인하세요. 3) 먼저 수수료로 쓸 ETH를 조금 받아야 해요.

핵심 정리

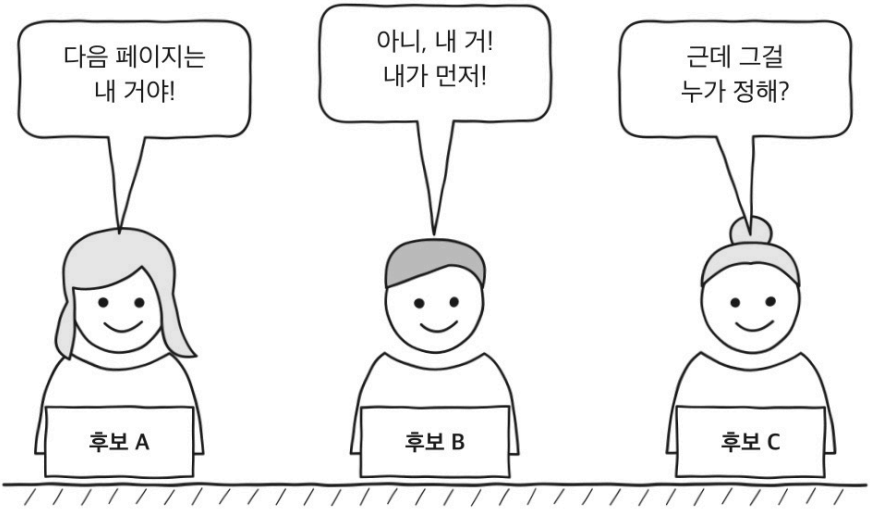
- ‘보냈다’와 ‘받았다’ 사이에는 길이 있어요. 노드, 대기열, 블록, 컨펌.
- 수수료는 블록의 자리를 사는 것이고, 금액과는 상관없어요.
- 결제란 올바른 네트워크에서 올바른 코인이 성공적으로 송금되고 컨펌이 충분히 쌓인 것을 말해요.

다음: 8장. 블록은 누가 모으고, 모두는 어떻게 바로 이 체인이 맞다고 동의할까요?

제8장. 네트워크는 어떻게 합의할까

다음 페이지는 누가 쓸까?

공용 장부에는 대장이 없어요. 수천 명의 참가자가 사본을 가지고 있고, 누구나 자기가 만든 다음 페이지를 내놓을 수 있습니다. 어느 것을 진짜로 칠까요?



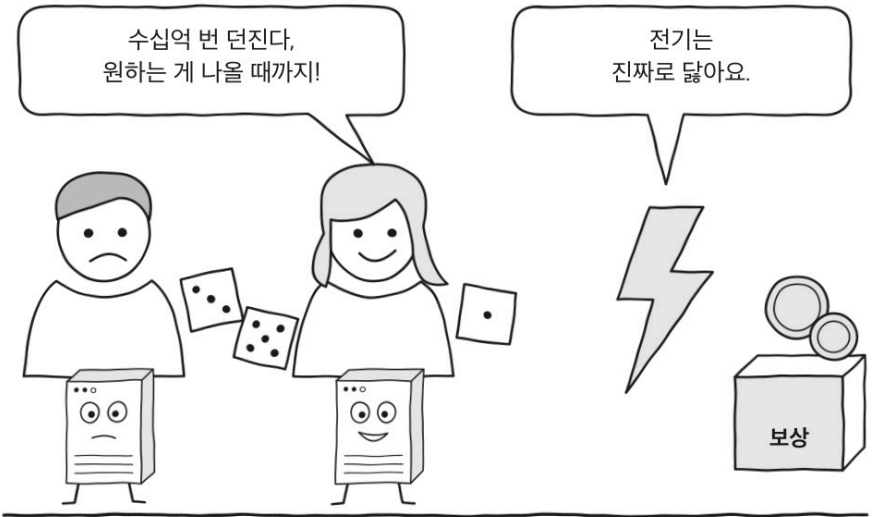
알아보기: 합의

참가자들이 하나의 답에 이르는 규칙을 합의(콘센서스)라고 해요. 이 규칙의 목표는 정직하게 참여하는 것보다 속이는 것이 더 비싸게 만드는 거예요.

단순히 표를 세는 것은 안 돼요. 사기꾼이 프로그램을 백만 개 띄우면 백만 표를 얻으니까요. 그래서 한 표에는 대가가 있어야 합니다. 대표적인 방법이 두 가지예요.

방법 1: 작업 증명

영어로 Proof of Work, 그 과정은 채굴(마이닝)이라고 불러요. 참가자들은 머리로는 풀 수 없고 하나하나 대입해야만 풀리는 문제를 두고 경주해요. 드문 조합이 나올 때까지 주사위를 던지는 것과 비슷합니다.



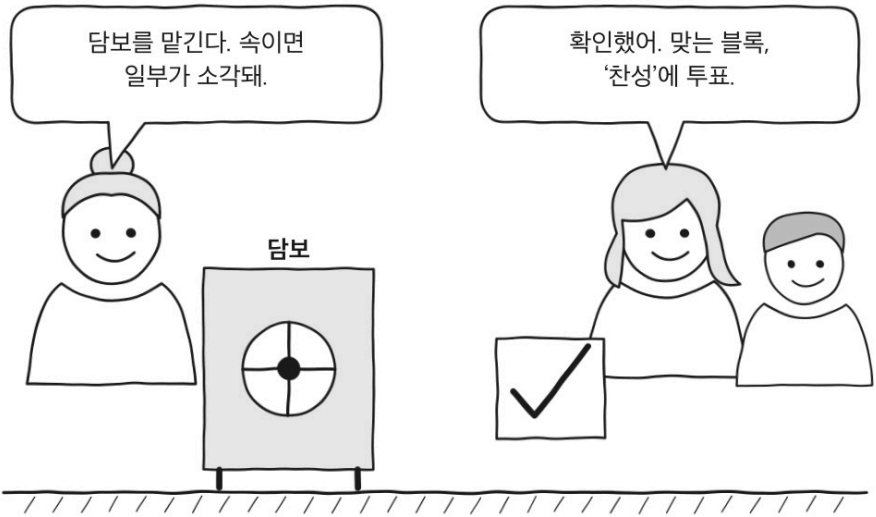
작업 증명

이긴 사람이 블록을 붙이고 보상을 받아요. 새 코인과 수수료입니다. 답을 확인하기는 쉽지만 찾기는 비싸요. Bitcoin이 이렇게 돌아갑니다.

- 블록은 평균 10분에 하나씩 나와요.
- 2016블록마다 문제의 난이도를 조정해 이 속도를 유지해요.
- 지금은 블록 하나에 새 코인 3.125 BTC를 주고, 보상은 약 4년마다 절반으로 줄어요.

방법 2: 지분 증명

영어로 Proof of Stake, 흔히 스테이킹이라고 해요. 참가자는 전기 대신 코인을 담보로 맡깁니다. 이런 참가자를 검증자(밸리데이터)라고 불러요. 검증자들이 차례로 블록을 제안하고, 나머지는 확인하고 투표합니다.



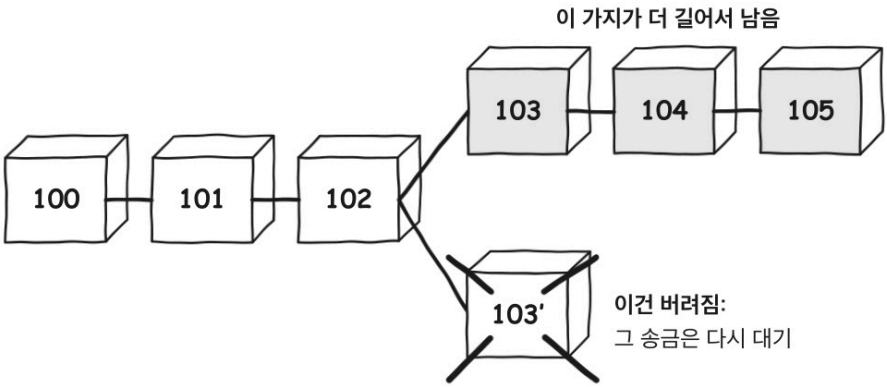
지분 증명

심각한 위반을 하면 담보의 일부가 소각되는데, 이것을 슬래싱이라고 해요. Ethereum은 2022년 9월 15일에 이 방식으로 바꿨고, 네트워크의 전력 소비가 약 99.95% 줄었습니다. 직접 검증자가 되려면 32 ETH가 필요해요.

변형도 있어요. TRON에서는 코인 보유자들이 뽑은 대표 27명이 블록을 씁니다. BNB Smart Chain에서도 정해진 수의 검증자들이 일해요.

페이지가 한꺼번에 두 장 나오면

네트워크는 크고, 소식은 순식간에 퍼지지 않아요. 같은 번호의 블록이 두 개 생겨 체인이 잠시 갈라지기도 합니다. 그러다 한쪽 가지가 이기고 다른 쪽은 버려져요. 이것을 체인 재구성(리오그, reorg)이라고 합니다.



버려진 블록의 송금이 흔적 없이 사라지는 것은 아니에요. 보통은 대기열로 돌아가 다른 블록에 들어갑니다. 하지만 못 들어갈 수도 있어요. 5장의 컨펌이 필요한 이유가 바로 이것입니다.

그림에서는 긴 가지가 이겨요. 정확히 말하면 Bitcoin은 더 많은 작업이 투입된 가지를, 스테이킹 네트워크는 담보의 표를 더 많이 받은 가지를 고릅니다.

파이널리티: 이제는 뒤집히지 않는다

파이널리티(최종 확정)는 그 뒤로는 블록이 바뀌지 않는 시점이에요. 어떤 네트워크는 이를 명시적으로 알리고, 어떤 네트워크는 블록이 쌓일수록 확신이 커질 뿐입니다.

네트워크	기록이 최종 확정됐다고 보는 대략의 시점
Bitcoin	명시적인 표시가 없음. 큰 금액은 6컨펌을 권하며, 약 1시간
Ethereum	보통 13–15분, 가끔 더 걸림
TRON	보통 약 1분
BNB Smart Chain	약 1초 이하
Solana	2026년 업그레이드 이후 1초도 안 걸리도록 설계됨
TON	약 1초

2026년 11월 27일 기준 각 네트워크 문서의 자료예요. 이 숫자들은 업그레이드에 따라 바뀝니다.

직접 해보기: 세 가지 '신선도' 묻기

Ethereum 노드에게는 가장 새 블록만 달라고 할 수 있는 게 아니에요. 1장의 `latest`에게는 친척이 둘 있어요. `safe`(이미 믿을 만함)와 `finalized`(최종 확정)입니다.

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

`finalized`를 `safe`와 `latest`로 바꿔 가며 블록 번호를 비교해 보세요. 이 장을 쓰던 시점에 노드는 이렇게 대답했어요.

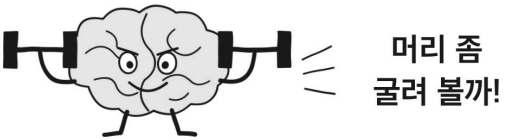
무엇을 물었나	블록 번호	최신 블록과의 차이
`latest`	26,502,104	—
`safe`	26,502,070	34블록, 약 7분
`finalized`	26,502,038	66블록, 약 13분

함정: “네트워크는 다 똑같아”

“컨펌 하나만 기다리면 끝”은 위험한 규칙이에요. 컨펌 하나가 보장하는 안전성은 네트워크마다 전혀 다릅니다. 네트워크와 금액을 보고 정하세요.

- 비싼 주문일수록 오래 기다려요.
- 큰 금액은 파이널리티가 있는 곳이라면 최종 확정까지 기다려요.
- 갓 나온 블록의 송금만 보고 되돌려받을 수 없는 것을 넘겨주지 마세요.

생각해 보기



1. 공용 장부에서는 왜 “프로그램 하나에 한 표” 원칙으로 투표할 수 없을까요?
2. Bitcoin 블록은 10분마다 나와요. 6컨펌을 기다리려면 평균 얼마나 걸릴까요? 3컨펌은요?
3. 한 손님은 커피 한 잔을 포장해 가고, 다른 손님은 원두 200kg을 사 가요. 아르티옴은 누구에게 파이널리티를 기다려야 할까요?

정답: 1) 프로그램은 공짜라 얼마든지 띄울 수 있어요. 2) 약 1시간, 약 30분. 3) 200kg을 사는 손님.

핵심 정리

- 합의는 대장 없는 참가자들이 하나의 체인을 고르는 규칙이에요.
- 한 표의 대가는 작업(채굴) 아니면 담보(스테이킹)예요.
- 갓 나온 블록은 바뀔 수 있어요. 최종 확정은 나중에, 네트워크마다 다르게 찾아와요.

다음: 9장. 코인, 토큰, 스마트 컨트랙트. USDT는 왜 ETH와 같지 않을까요.

제9장. 코인, 토큰, 스마트 컨트랙트

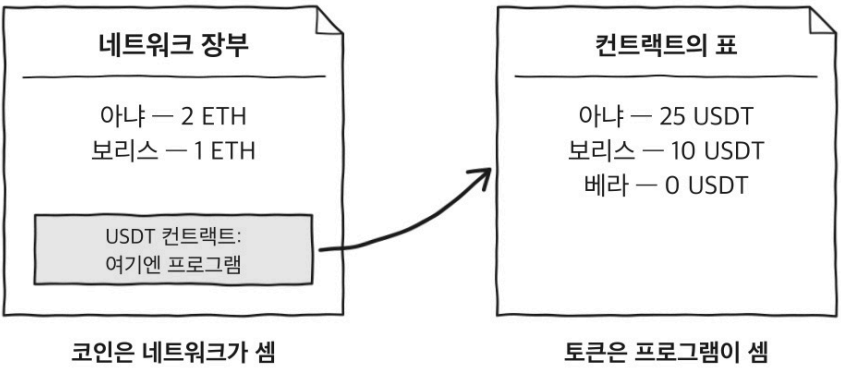
“돈 냈는데 그쪽엔 없대요”

손님이 Ethereum 네트워크에서 25 USDT를 보냈어요. 아르티옴이 1장처럼 노드에게 물었는데 잔액이 그대로예요. 돈이 사라진 걸까요? 아니에요. 다른 방식으로 물었어야 해요.

알아보기: 코인과 토큰

기본 코인은 네트워크가 직접 관리하는 코인이에요. Ethereum의 ETH, TRON의 TRX, BNB Smart Chain의 BNB죠. 수수료도 이것으로 냅니다.

토큰은 네트워크 안에 사는 프로그램이 관리하는 기록이에요. USDT는 토큰입니다.



	기본 코인	토큰
누가 기록을 관리하나	네트워크 자체	스마트 컨트랙트
Ethereum의 예	ETH	USDT, USDC
수수료는 무엇으로 내나	그 코인 자체	네트워크의 기본 코인
잔액은 어떻게 묻나	`eth_getBalance`	컨트랙트에 `eth_call`

네트워크마다 토큰의 구조는 달라요. 여기서는 Ethereum과 그와 비슷한 네트워크를 봅니다.

스마트 컨트랙트: 점원 없는 자판기

스마트 컨트랙트는 블록체인에 적힌 프로그램이에요. 자판기와 비슷합니다. 넣고 누르면, 미리 정해진 규칙에 따라 결과가 나와요. 모든 노드가 똑같이 실행합니다.



- 서명이 담긴 계약서가 아니라 말 그대로 프로그램이에요.
- 보통은 바꿀 수 없지만, 어떤 만든이는 바꿀 수 있는 길을 남겨 둬요.
- 프로그램에는 특별한 권한을 가진 '주인'이 있을 수 있어요.
- 프로그램의 오류는 돈 손실로 이어질 수 있고, 되돌려 줄 사람이 없어요.

직접 해보기: 토큰 잔액

`eth_call` 메서드는 "아무것도 바꾸지 말고 컨트랙트에게 물어봐"라는 뜻이에요. 장부에 아무것도 쓰지 않기 때문에 네트워크는 이런 질문에 수수료를 받지 않습니다. USDT 컨트랙트에게, 컨트랙트 자신의 주소에 USDT가 얼마나 있는지 물어볼게요.

```
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` 는 어느 컨트랙트에게 묻는지예요. `data` 는 세 조각을 이어 붙인 것입니다.

조각	뜻
`0x70a08231`	어떤 질문인가: "잔액이 얼마야?" (`balanceOf`)
0 24개	필요한 길이까지 채우기
40자	누구의 잔액인가: `0x` 를 뺀 주소

이 장을 쓰던 시점에 노드는 이렇게 대답했어요.

```
{"jsonrpc":"2.0","id":1,"result":"0x0000000000000000000000000000000000000000000000000000014d0a9aa6c5"}
```

0들 뒤에 있는 `14d0a9aa6c5`는 1,430,402,016,965예요. Ethereum의 USDT는 소수점 아래가 여섯 자리이니, 컨트랙트 주소에는 1,430,402.016965 USDT가 있는 거예요.

이 돈은 어디서 왔을까요? 정황상 대부분은 '잘못 보낸' 송금이에요. 받는 사람 주소 자리에 컨트랙트 주소를 붙여 넣은 거죠. 보낸 사람이 되찾아 올 평범한 방법은 없습니다.

소수점은 어디에 찍을까

컨트랙트는 정수만 저장해요. 몇 자리를 소수점 아래로 떼어 낼지는 `decimals`라는 별도의 속성이 알려 줍니다.

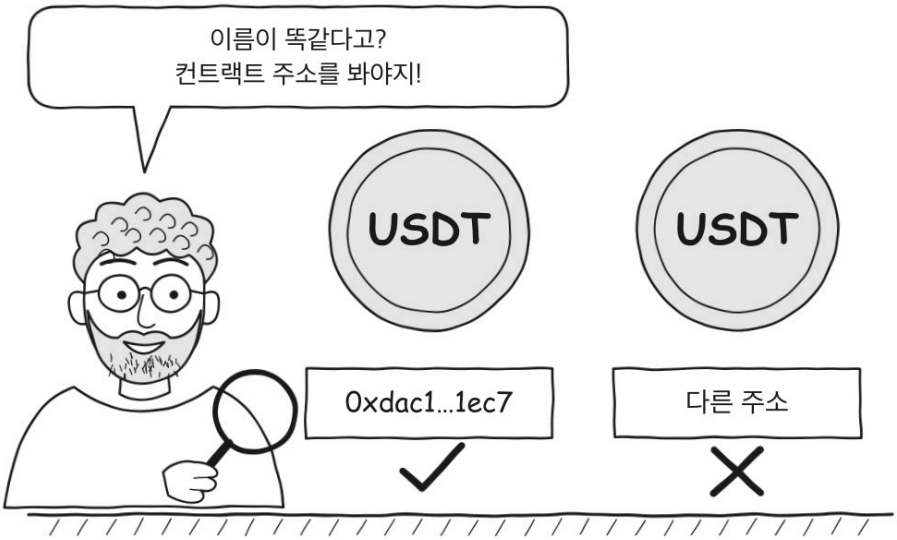
무엇	소수점 아래 자릿수
ETH, 기본 코인	18
Ethereum의 USDT	6
BNB Smart Chain의 'Tether USD'(USDT) 토큰	18

이름은 같은데 자릿수는 세 배예요. 헷갈리면 1조 배 틀리게 됩니다. 게다가 BNB Smart Chain의 이 토큰 주소는 Tether 회사 사이트의 USDT 컨트랙트 목록에 없어요. 이름이 같다고 같은 발행물은 아닙니다.

2026년 11월 27일에 공개 노드 요청과 tether.to/en/supported-protocols 페이지로 확인했어요.

함정: 가짜 토큰

누구든 자기 토큰에 'USDT'라는 이름을 붙일 수 있어요. 이름은 아무것도 증명하지 않습니다. 진짜 USDT는 네트워크와 컨트랙트 주소의 짝으로 알아봐요.



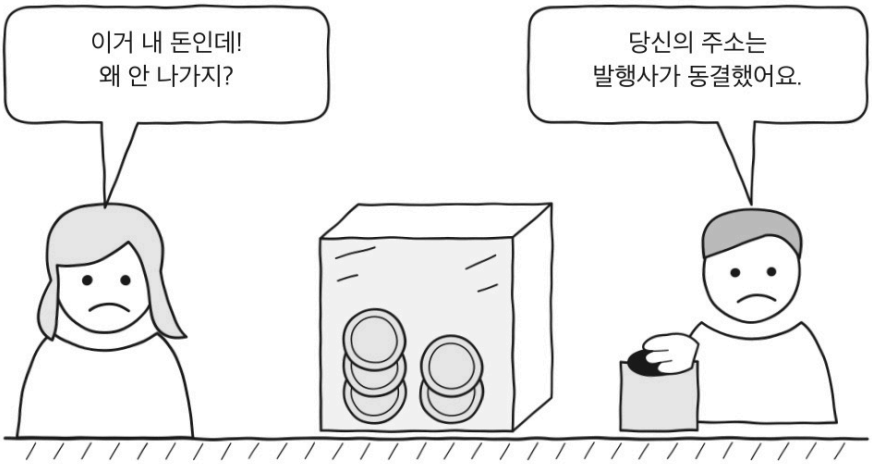
그래서 가게는 모든 것을 한꺼번에 확인해요.

- 네트워크
- 토큰 컨트랙트 주소
- 받는 사람
- 올바른 자릿수를 적용한 금액
- 성공 여부와 컨펌

스테이블코인: 편하지만 주인이 있다

스테이블코인은 1달러 언저리를 유지하려고 해요. USDT의 경우 이것은 자연법칙이 아니라 발행 회사의 약속입니다.

발행사에게는 권한도 있어요. Tether의 이용 약관에는 토큰을 동결할 권리가 분명히 적혀 있습니다(2026년 2월 26일 개정판, tether.to/en/legal).



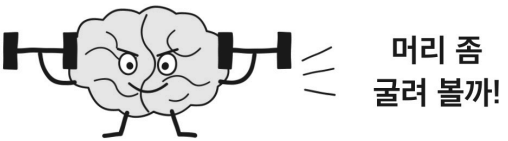
코인은 보이는데 안 움직임

두 번째 함정: “쓸 수 있게 허락해 주세요”

웹사이트가 허용(`approve`)에 서명해 달라고 할 수 있어요. “이 컨트랙트가 내 토큰을 써도 된다”라는 뜻이에요. 사기꾼에게 금액 제한 없이 허용하면, 나중에 새 서명 없이도 토큰을 가져갑니다. 시드 문구를 알 필요도 없어요.

> 원칙: 무엇에 서명하는지 모르겠다면, 서명하지 마세요.

생각해 보기



1. Ethereum의 USDT에 대해 노드가 25,000,000이라는 숫자를 돌려줬어요. USDT로 얼마일까요?
2. 손님이 이름은 USDT이지만 컨트랙트 주소가 다른 토큰을 보냈어요. 주문은 결제된 걸까요?
3. 아르티옴에게 Ethereum의 USDT 100개와 ETH 0개가 있어요. 공급업체에게 USDT를 보낼 수 있을까요?

정답: 1) 25. 2) 아니요. 3) 아니요. 수수료를 낼 것이 없어요. 7장을 보세요.

핵심 정리

- 코인은 네트워크가, 토큰은 스마트 컨트랙트가 관리해요.
- 토큰은 이름이 아니라 네트워크와 컨트랙트 주소로 알아봐요.
- 금액은 소수점 자릿수를 고려해서 읽어요.

파트 2 끝. 아르티옴은 블록, 키, 수수료, 토큰이 무엇인지 알게 됐어요. 하지만 노드는 아직 그에게 블랙박스예요. 다음: 파트 3, 10장. 노드에는 어떤 종류가 있을까요.

제10장. 노드의 종류

“그때는 얼마가 있었지?”

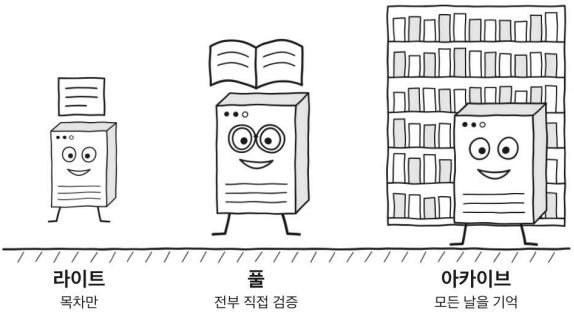
아르티옴은 장부 정리를 하다가 몇 달 전에 가게 주소에 얼마가 있었는지 알고 싶어졌어요. 노드는 오늘에 대해서는 척척 대답하는데, 과거에 대해서는 거절해요. 블록체인은 모든 것을 보관하는 게 아니었나요?

알아보기: 기록과 잔액

여기에는 서로 다른 두 가지가 숨어 있어요.

- 히스토리: 기록 그 자체. 블록과 송금이에요.
- 상태: 특정 순간에 누가 무엇을 가졌는지. 잔액과 컨트랙트의 데이터예요.

은행과 비슷해요. 거래 내역은 있지만, 작년의 매 분마다 잔액이 따로 저장돼 있지는 않아요. 다시 계산해야 하죠. 노드들은 이것들을 얼마나 많이 기억하는지가 서로 다릅니다.

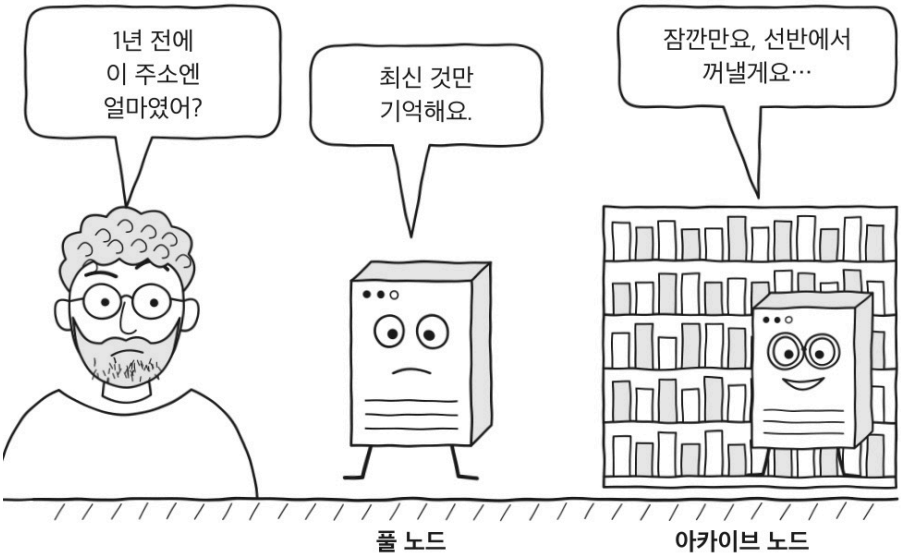


대표적인 세 가지

풀 노드는 네트워크 규칙에 따라 블록을 검증하고 최근 상태를 보관해요. Ethereum에서는 보통 최근 128블록 정도, 즉 약 25분입니다. 정확한 깊이는 프로그램과 설정에 따라 달라요.

아카이브 노드는 맨 처음부터 모든 것을 검증하고, 받은 것을 지우지 않아요. 어느 블록 시점의 상태든 기억하는 대신 디스크가 훨씬 많이 필요합니다.

라이트 노드는 블록 헤더만 내려받고, 자세한 내용은 다른 노드에게 물어요. 좋은 생각이지만, 2026년 11월 27일 현재 Ethereum에는 일상적으로 쓸 만큼 준비된 라이트 노드가 없습니다.



프루닝 노드: 검증하고 버린다

Bitcoin에는 디스크를 아끼는 모드가 있어요. 프루닝 노드(pruned)는 모든 것을 검증한 뒤 오래된 블록을 지웁니다. 여전히 어엿한 풀 노드이고, 다만 지운 것은 내줄 수 없을 뿐이에요. Bitcoin Core 개발자들에 따르면 10GB로도 충분합니다.

Ethereum에도 비슷한 기능이 생겼어요. 2025년 7월부터 노드 프로그램들은 가장 오래된 블록, 즉 2022년 9월 스테이킹 전환 전에 나온 블록의 내용을 저장하지 않을 수 있어요. 설정 사항이고, 모든 곳에서 켜져 있지는 않아요. 그래서 '풀' 노드라도 아주 오래된 기록은 기억하지 못할 수 있습니다.

직접 해보기: 과거에 대해 묻기

같은 주소의 잔액을 두 번 물어볼게요. 먼저 지금 기준입니다.

```
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` 는 42 wei, 정말 부스러기예요. 이제 `"latest"` 를 `"0xe4e1c0"` , 즉 15,000,000번 블록으로 바꿔 보세요. 공개 노드는 이렇게 대답합니다.

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

우리말로 하면 “공개 엔드포인트에서는 아카이브 데이터를 쓸 수 없으니, 개인 엔드포인트를 쓰라”는 뜻이에요. 고장이 아니라 무료 서비스의 경계입니다. RpcNode에서는 유료 요금제에 아카이브 접근이 있고, Free에는 없어요(3장).

무게는 얼마나 될까

노드	대략의 디스크
Bitcoin, 프루닝	10GB부터
Bitcoin, 풀	약 900GB
Ethereum, 풀	약 2TB 이상
Ethereum, 아카이브	프로그램에 따라 2TB에서 14TB 이상

Ethereum 자료는 2026년 11월 27일 기준 ethereum.org의 노드 실행 안내 페이지에서 가져왔어요.

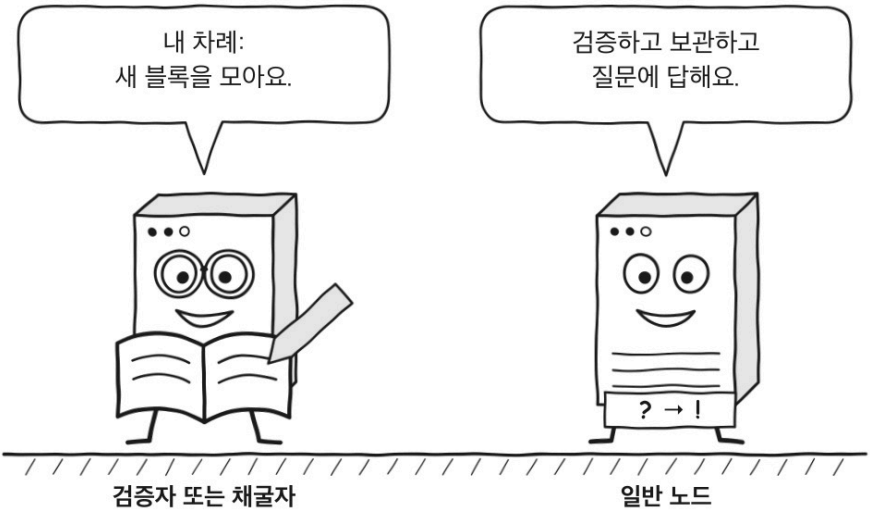
두 번째 줄은 직접 확인할 수 있어요. Bitcoin 노드는 자기 크기를 솔직하게 알려 줍니다.

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

긴 응답에서 두 줄을 찾아보세요. 이 장을 쓰던 시점에는 `"size_on_disk":896027659764`(보조 파일까지 합쳐 약 896GB)와 `"pruned":false`, 즉 프루닝하지 않은 노드였어요.

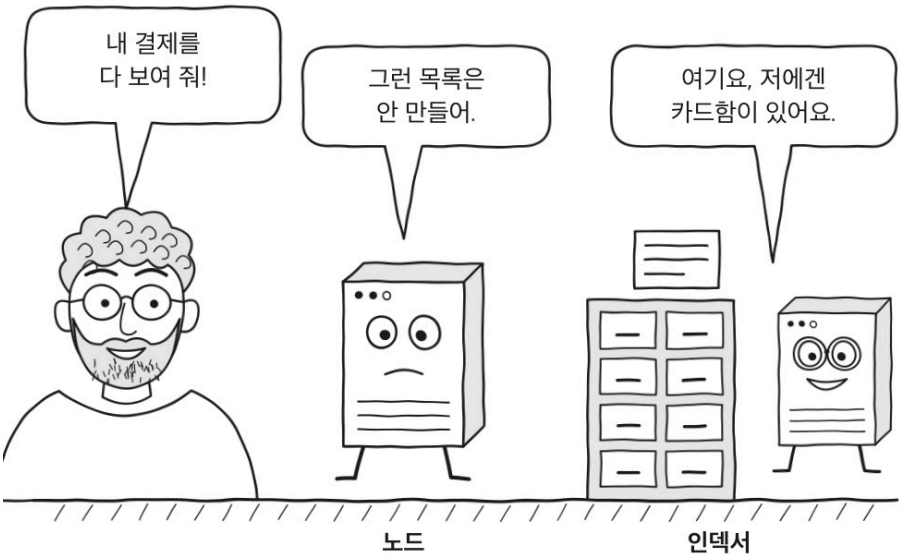
그럼 블록은 누가 쓸까?

그건 별도의 역할이에요. 검증자나 채굴자가 블록 생성에 참여합니다(8장). 결제를 확인하는 데 그럴 필요는 없어요. 읽고, 검증하고, 대답하는 노드면 충분합니다.



함정: "내 결제를 전부 노드에게 물어보지 뭐"

Ethereum 노드의 표준 질문 목록에는 "이 주소의 모든 송금을 보여 줘"가 없어요. 아카이브 노드도 그건 못 해요. `eth_getTransactionCount` 메서드는 그 주소에서 보낸 트랜잭션의 개수만 알려 줍니다.

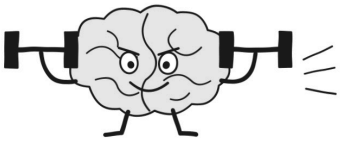


그런 목록에는 인덱서가 필요해요. 블록을 읽어서 기록을 카드함에 정리해 두는 프로그램입니다. 블록체인 탐색기 사이트도 이렇게 동작해요. 가게라면 처음부터 자기 결제를 기록해 두고 새 결제를 지켜보는 편이 쉬워요. 이 이야기는 26장에 있습니다.

내 일에는 무엇이 필요할까

할 일	알맞은 것
최근 결제 확인	일반 풀 노드
과거 특정 날짜의 잔액 알기	아카이브 노드
한 주소의 모든 송금 목록 얻기	인덱서
블록 생성에 참여하기	검증자 또는 채굴자

생각해 보기



머리 좀 굴려 볼까!

1. 아르티옴이 하루에 점 하나씩 찍어 1년 치 가게 잔액 그래프를 그리려 해요. 어떤 노드가 필요할까요?
2. Bitcoin 프루닝 노드는 옛 블록을 지웠어요. 새 블록 검증을 맡겨도 될까요?
3. 손님에게 지난 주문과 결제를 모두 보여 주려면 아카이브 노드와 가게 자체 데이터베이스 중 무엇이 필요할까요?

정답: 1) 아카이브 노드. 2) 네, 지우기 전에 모두 검증했어요. 3) 자체 데이터베이스. 노드는 그런 목록을 관리하지 않아요.

핵심 정리

- 히스토리는 기록이고, 상태는 특정 순간의 잔액이에요.
- 풀 노드는 최근 상태를, 아카이브 노드는 어느 시점의 상태든 기억해요.
- 노드는 주소의 송금 목록을 주지 않아요. 그건 인덱서의 일이에요.

다음: 11장. 우리는 벌써 노드에게 질문을 열 번쯤 했어요. 이제 어떤 말로 했는지 알아볼 때예요. JSON-RPC, REST, gRPC, WebSocket입니다.

제11장. 노드와 대화하는 법

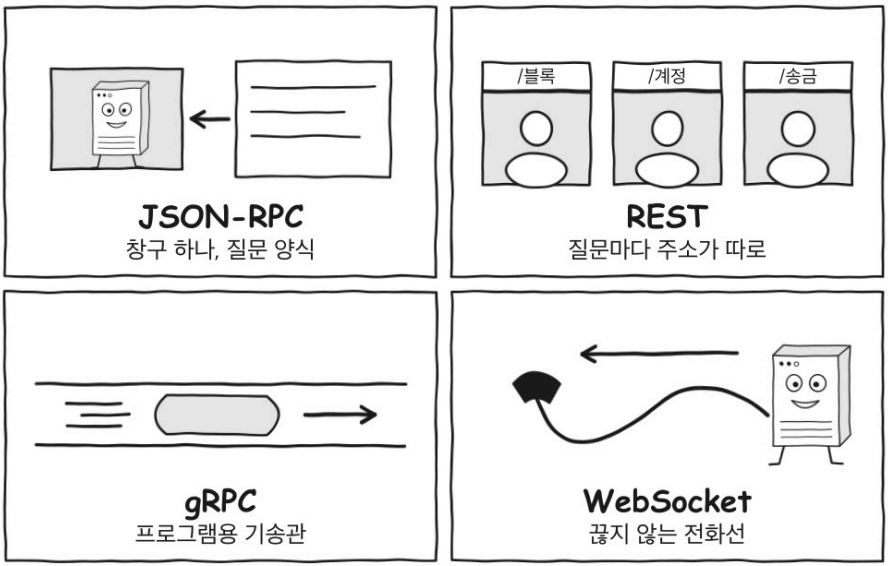
요청에 적힌 이 글자들은 뭘까?

아르티옴은 이 책의 명령을 벌써 열 번쯤 복사해서 썼고, 잘 돌아갔어요. 이제 그 안에 무엇이 적혀 있는지 알아볼 때입니다. RPC란 무엇인지, 괄호는 왜 이렇게 많은지, 네트워크마다 문서의 요청 모양은 왜 다른지요.

알아보기: API와 RPC

API는 한 프로그램이 미리 정해진 규칙에 따라 다른 프로그램에게 말을 거는 창구예요.
RPC(원격 프로시저 호출)는 다른 컴퓨터의 프로그램에게, 마치 옆에 있는 것처럼 무언가를 해 달라고 부탁하는 방식입니다. 노드는 이런 부탁을 받아요.

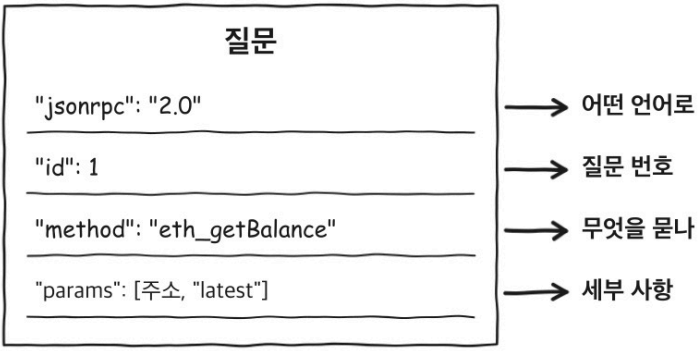
대화 방식은 몇 가지가 있어요.



방식	생김새	어디서 만나나
JSON-RPC	주소는 하나, 요청 안에 메서드와 매개변수를 적음	Ethereum 계열, Solana, Bitcoin
REST	질문마다 주소 안의 경로가 따로 있음	TRON, TON, Aptos
gRPC	엄격한 스키마를 따르는 압축된 이진 메시지	프로그램끼리의 데이터 교환, 예를 들어 Sui
WebSocket	닫히지 않는 연결	노드가 새 소식을 스스로 알려야 할 때

WebSocket은 언어가 아니라 통신선이에요. Ethereum에서는 이 선으로 똑같은 JSON-RPC 메시지가 오갑니다. 어떤 네트워크는 여러 방식을 한꺼번에 지원해요.

JSON-RPC 뜯어보기



JSON은 데이터를 적는 간단한 방법이에요. 중괄호, 따옴표로 감싼 이름, 그리고 값이죠. 3장의 명령을 줄 단위로 뜯어보면 이렇습니다.

명령의 조각	뜻
`curl`	터미널에서 요청을 보내는 프로그램
`https://…`	엔드포인트 주소
`-H '…'`	"안에 든 것은 JSON"이라는 표시
`-d '{…}'`	질문 그 자체

응답에서는 `result`(결과)와 `error`(실패 내용)를 봐요. 어떤 노드는 둘 중 한 필드만 보내고, 어떤 노드는 둘 다 보내는데 이때 빈 쪽은 `null` 입니다. 응답의 `id`는 질문의 것과 같아서, 이것으로 짝을 맞춰요.

한 번에 여러 질문하기

질문들을 대괄호 안의 목록으로 묶을 수 있어요. 이것을 배치(batch)라고 합니다.

```
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"}]
```

첫 번째 응답은 블록 번호 26,502,176이고, 두 번째는 네트워크 번호예요. Ethereum은 1번입니다. JSON-RPC 규칙상 배치의 응답은 어떤 순서로든 올 수 있으니 `id`를 보세요. 그리고 기억하세요. 배치는 왕복 횟수는 줄여 주지만 한도는 줄여 주지 않아요. RpcNode에서 질문 열 개짜리 배치는 요청 열 건으로 계산되고, 통째로 받아들여지거나 통째로 거절됩니다.

이번에는 REST

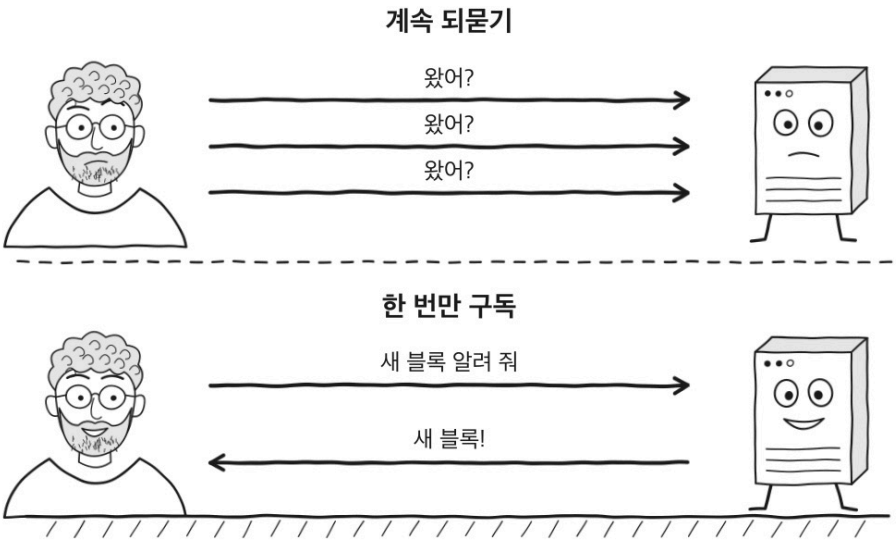
여기서는 질문이 주소 안에 들어 있어요. TRON 네트워크에 최신 블록을 물어볼게요.

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

송금이 담긴 블록에 관한 긴 응답이 옵니다. 이 장을 쓰던 시점에는 88,351,889번 블록이었고, 트랜잭션이 194건 들어 있었어요. 다른 질문은 주소 뒤의 다른 경로로 합니다.

WebSocket: 묻지 말고 듣기

1초마다 "왔어?"라고 묻는 것은 낭비예요. 열린 통신선에서는 한 번만 구독하면 노드가 알아서 소식을 보내 줍니다.



RpcNode의 공개 노드는 WebSocket에 응답하지 않아서, 개인 엔드포인트가 필요해요. 그 주소는 `wss://`로 시작합니다. JavaScript로 대강 그려 보면 이렇습니다.

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

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('연결이 끊어졌어요')
```

선은 끊어지기 마련이에요. 프로그램은 다시 접속하고, 다시 구독하고, 놓친 것이 없는지 확인할 수 있어야 합니다. 구독도 크레딧을 써요.

gRPC

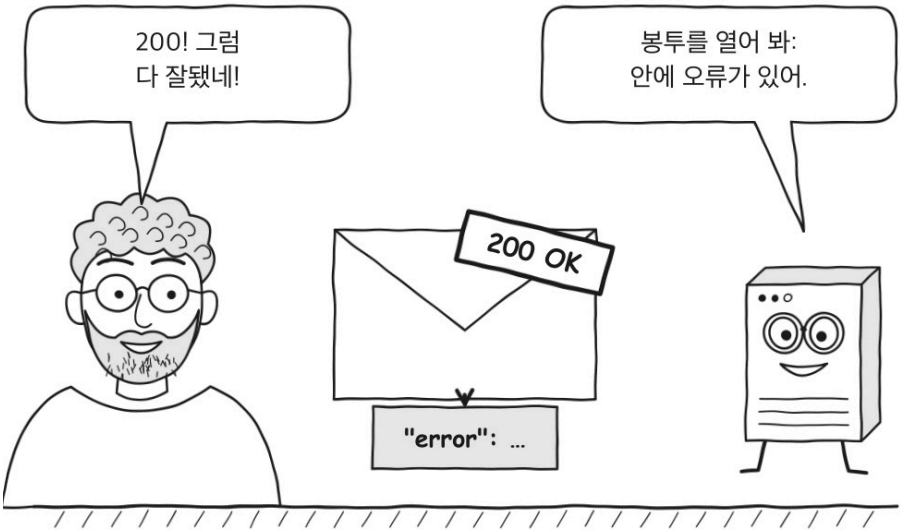
속도와 엄격한 형식이 중요한 프로그램을 위한 방식이에요. 메시지는 눈으로 읽을 수 없고, 필드는 별도의 파일에 미리 정의해 둡니다. 막 시작하는 가게에는 보통 필요 없어요. RpcNode에서 gRPC는 모든 네트워크에 있는 것이 아니니, 엔드포인트의 전송 방식 목록을 확인하세요.

함정: “200이면 다 괜찮은 거지”

200은 배달 코드예요. 봉투가 도착했다는 뜻이죠. 그 안에는 오류가 들어 있을 수 있어요. “hello”라는 주소의 잔액을 물어볼게요.

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

응답 코드는 200인데, 그 안에 `-32602` 번 `error` 가 있어요. 잘못된 매개변수라는 뜻입니다.



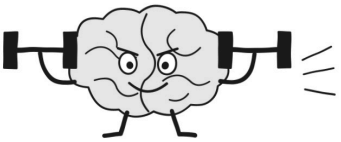
그래서 프로그램은 언제나 코드와 `error` 필드를 모두 확인해요.

자주 만나는 오류

코드	무슨 일인가	할 일
`-32700`	요청 글을 해석할 수 없음	괄호와 따옴표 확인
`-32600`	요청이 규칙에 맞지 않음	필드 확인
`-32601`	메서드를 찾을 수 없거나 쓸 수 없음	이름과 요금제 확인
`-32602`	잘못된 매개변수	값 확인
`-32005` (HTTP 429)	RpcNode: 너무 자주 요청함	기다렸다가 덜 자주 묻기
`-32006` (HTTP 402)	RpcNode: 크레딧 부족	사용량과 요금제 확인

처음 네 코드는 JSON-RPC 2.0 표준에서, 마지막 두 개는 RpcNode의 코드예요.

생각해 보기



머리 좀 굴려 볼까!

1. 배치의 첫 번째 요청에 `eth_blockNumber` 대신 `eth_blockNumbr`라고 써 보세요. 어떤 코드가 올까요? 올바른 두 번째 질문에는 대답할까요?
2. 저절로 갱신돼야 하는 '마지막 블록' 전광판에는 1초마다 묻기와 구독 중 무엇이 좋을까요?
3. 프로그램이 HTTP 200을 받자마자 주문을 결제 완료로 표시했어요. 무엇이 문제일까요?

정답: 1) `-32601`. 게다가 전체에 하나만 옵니다. RpcNode는 배치 전체를 거절해서 두 번째 질문도 답을 받지 못해요. 2) 구독. 3) `error` 필드도, 송금 자체도 확인하지 않았어요(7장).

핵심 정리

- JSON-RPC는 메서드를 요청 안에, REST는 주소 안에 적어요.
- WebSocket은 선을 열어 두어 노드가 스스로 소식을 전하게 해요.
- 성공이란 200 코드가 아니라, `error`가 없고 기대한 내용이 담긴 응답이에요.

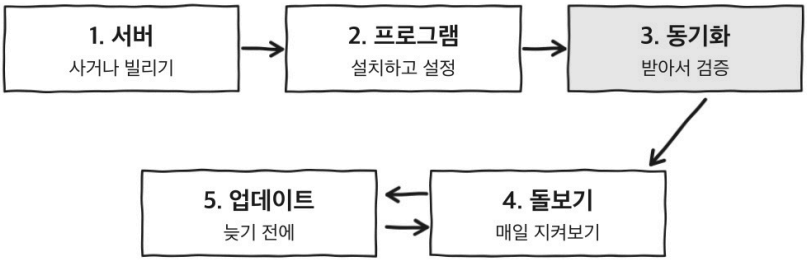
다음: 12장. 창구 너머를 들여다봅시다. 내 노드는 어떻게 살아가고, 동기화란 무엇이며, 왜 고장이 날까요.

제12장. 내 노드의 속사정: 설치, 동기화, 고장

"그거 하룻저녁이면 돼요!"

아는 개발자가 아르티옴을 설득해요. "내려받아서 실행하면 내 노드가 생긴다니까요." 실행하는 데는 정말 오래 걸리지 않아요. 오래 걸리는 건 그다음부터입니다.

알아보기: 노드의 일생

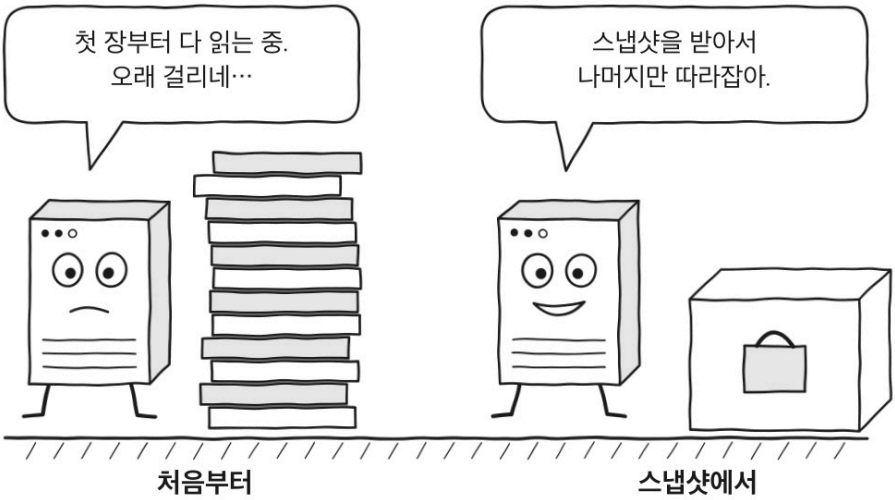


4, 5단계는 노드가 사는 동안 반복

1. 서버. 요구 사양은 2장에서 봤어요.
2. 프로그램. 네트워크마다 따로 있고, 여러 개 중에 고를 수 있는 경우도 많아요. Ethereum에는 두 개가 함께 필요합니다. 실행 클라이언트(송금과 컨트랙트를 실행)와 합의 클라이언트(어느 체인이 맞는지 감시)예요.
3. 동기화. 노드가 네트워크를 따라잡아요.
4. 돌보기. 뒤처지지 않았는지, 공간은 충분한지 살펴요.
5. 업데이트. 네트워크가 규칙을 바꾸면 미리 프로그램을 바꿔요.

동기화: 네트워크 따라잡기

새 노드는 아무것도 몰라요. 다른 노드들을 따라잡기 전까지는 잔액을 물어봐야 소용없어요. 과거를 대답하니까요.



- 처음부터. 노드가 모든 블록을 직접 검증해요. 믿을 만하지만 오래 걸립니다. Bitcoin의 첫 다운로드는 약 740GB예요.
- 스냅샷에서. 만들어 둔 데이터베이스 사본을 가져와서 나머지를 따라잡아요. 빠르지만 스냅샷을 만든 사람을 믿어야 합니다.
- 중간 방식. Ethereum 프로그램들은 보통 최신 상태를 내려받은 뒤 블록 헤더와 대조해요.

시간은 몇 시간에서 여러 날까지 걸려요. 네트워크, 방식, 디스크, 회선에 따라 다릅니다.

직접 해보기: 두 줄짜리 설정

`bitcoin.conf` 설정 파일의 첫머리는 이런 모습일 수 있어요.

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

줄	뜻
`server=1`	RPC로 질문을 받기. 따로 허용하지 않으면 같은 컴퓨터에서 온 것만
`prune=5000`	프루닝 노드(10장): 옛 블록은 지우고 약 5GB만큼만 보관. 부가 데이터가 추가로 공간을 차지함

완전한 설치 안내가 아니에요. 설치하기 전에 사용하는 프로그램의 문서를 읽으세요.

노드가 따라잡았는지 아는 법

노드에게 직접 물어볼 수 있어요. Ethereum에서는 이렇게 합니다.

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

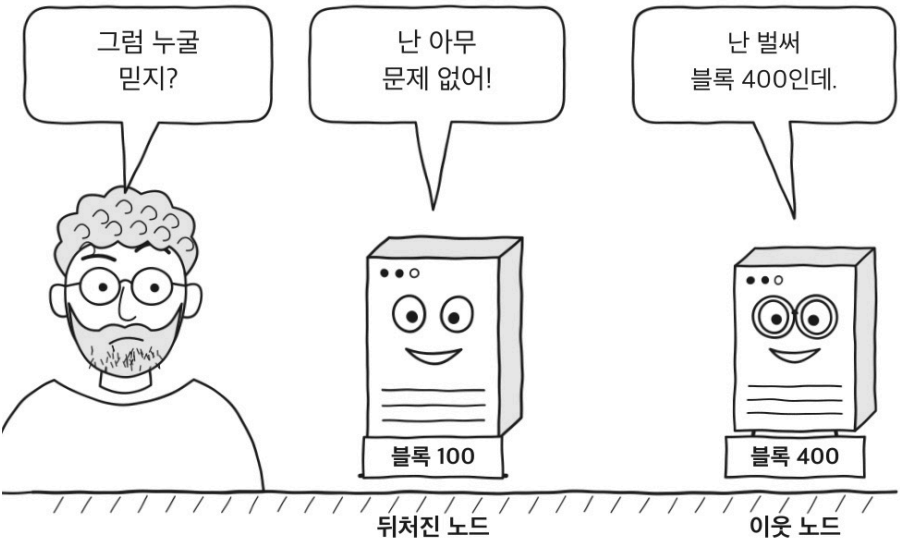
```
{"jsonrpc":"2.0","id":1,"result":false}
```

`false`는 "지금은 따라잡는 중이 아니야"라는 뜻이에요. 따라잡는 중이라면 대신 숫자들이 옵니다. 지금 어디까지 왔고 어디까지 가야 하는지요. Bitcoin에서는 10장의 `getblockchaininfo` 응답을 봐요.

필드	확인 당시 값	뜻
`blocks`	977,715	검증한 블록 수
`headers`	977,715	알고 있는 블록 수
`initialblockdownload`	`false`	첫 다운로드는 끝남

함정: "자기가 괜찮다고 했어"

노드는 자기가 아는 것을 바탕으로 판단해요. 다른 노드들과 연결이 끊기면, 뒤처진 줄도 모를 수 있습니다.



그래서 두 가지를 확인해요. 마지막 블록이 얼마나 새것인지와 다른 출처와의 비교입니다. 신선도는 쉽게 알 수 있어요. 마지막 블록의 시각을 달라고 하세요.

```
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"
```

이것은 1970년부터 흐른 초 수예요. 현재 시각에서 빼 보세요(여러분이 받은 숫자를 넣으세요).

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

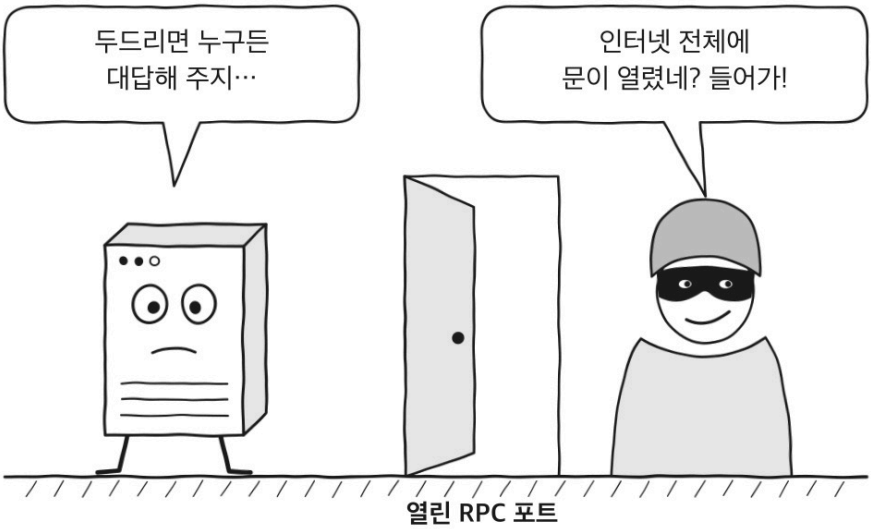
블록의 나이가 초 단위로 나와요. 이 장을 쓰던 시점에는 7초였어요. 12초마다 블록이 나오는 Ethereum에서는 정상입니다. 여러분의 노드에서 수천이 나온다면 뒤처진 거예요. 1장에서 한 시간 뒤처진 노드를 물었던 질문, 기억나죠?

무엇이 고장 날까

증상	흔한 원인	대처
노드가 멈춤	디스크 공간이 바닥남	공간을 지켜보고 미리 늘리기
계속 뒤처짐	느린 디스크나 회선, 적은 피어	장비와 네트워크 점검
네트워크 업그레이드 뒤 '멈춰 섬'	옛 버전 프로그램	공지된 날짜 전에 업데이트
정전 뒤 실행되지 않음	데이터베이스 손상	백업에서 복구하거나 다시 동기화
뚜렷한 이유 없이 느림	남의 요청이 부하를 줌	외부인의 접근 차단

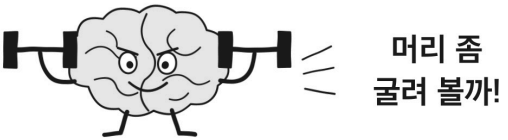
두 번째 함정: 열린 문

Bitcoin Core와 Geth는 처음부터 자기 컴퓨터에서 온 RPC 질문만 받아요. 괜히 그렇게 만든 게 아닙니다. 포트를 인터넷 전체에 열면 아무나 들이는 셈이고, 노드에 지갑이 있다면 돈까지 위험해져요.



- 내 주소에서만 접근하도록 허용하세요(방화벽).
- 노드 앞에 키 검사를 두세요.
- 사이트에 응답하는 노드에는 돈이 든 지갑을 두지 마세요.

생각해 보기



1. `eth_syncing`은 `false`라고 대답하는데 마지막 블록이 40분 전 것이에요. 노드는 건강할까요?
2. 2000GB 디스크에 1900GB를 썼고 하루에 5GB씩 늘어요. 며칠 뒤에 모든 게 멈출까요?
3. 사이트가 다른 서버에 있어요. "사이트가 물어볼 수 있게" RPC 포트를 모두에게 열어도 될까요?

정답: 1) 아니요. 2) 늦어도 20일 뒤. 고치기는 오늘 시작해야 해요. 3) 안 돼요. 사이트의 주소만 허용하세요.

핵심 정리

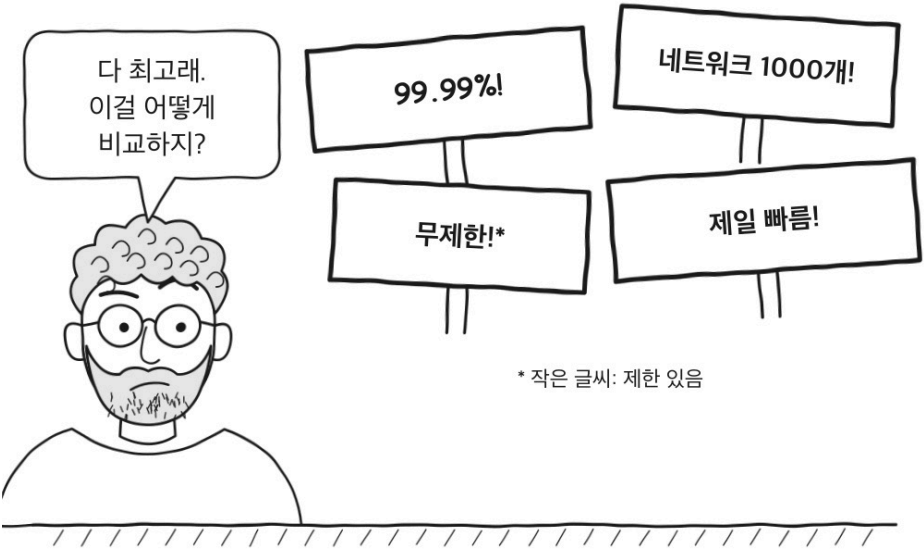
- 노드를 켜는 것은 일의 끝이 아니라 시작이에요.
- 건강은 블록의 신선도와 다른 출처와의 비교로 확인해요.
- RPC 포트는 인터넷 전체에 열지 않아요.

다음: 13장. 이 모든 것이 취미에 맞지 않는다면, 제공업체를 어떻게 고르고 무엇을 물어야 할까요.

제13장. 제공업체 고르는 법

다들 자기가 최고래요

아르티옴은 마음을 정했어요. 내 노드는 나랑 안 맞아. 제공업체 사이트들을 열어 보니 느낌표가 빽빽해요.



알아보기: 먼저 나부터

내게 무엇이 필요한지 모른 채 제공업체를 비교해 봐야 소용없어요. 먼저 대답해 보세요.

- 어떤 네트워크가 필요하고, 테스트넷도 필요한가?
- 초당, 그리고 월간 요청은 몇 건인가?
- 옛 기록(아카이브)이 필요한가?
- 사건을 곧바로 알아야 하는가(WebSocket, 알림)?

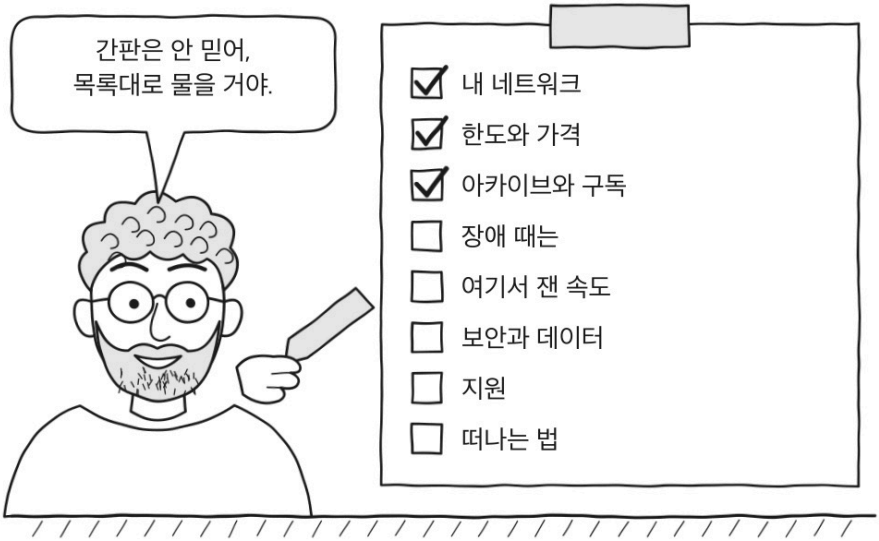
내 요청 수 세어 보기

아르티옴에게는 결제를 기다리는 주문이 20건 있고, 사이트는 각 주문을 10초에 한 번 확인해요.

> 주문 20건 × 분당 6회 = 분당 요청 120건, 즉 초당 2건.
> 30일 동안: 요청 5,184,000건.

여기에 메서드의 크레딧 가격을 곱하세요. 메서드마다 값이 달라요. 그리고 이렇게 자주 물어야 하는지도 생각해 보세요. 30초에 한 번이면 요청이 3분의 1로 줄어듭니다.

질문 목록



물어볼 것	경계할 신호
내 네트워크가 있나? 필요한 메서드도 있나?	메서드 목록 없이 길기만 한 네트워크 목록
한도: 초당 요청과 총량은? 넘으면 어떻게 되나?	숫자 없는 '무제한', 불분명한 추가 요금
아카이브, WebSocket, gRPC: 어느 요금제, 어느 네트워크에서?	"다 있어요"라면서 세부 내용이 없음
노드 장애: 다른 노드로 바꿔 주나? 서버는 어디에?	대답 없음
가용성: 무엇을 약속하고, 어떻게 재며, 어기면 어떻게 되나?	기간도 조건도 없는 퍼센트
보안: 주소와 사이트를 제한할 수 있나? 요청에 관해 무엇을 기록하나?	키를 아무나 써도 통함
지원: 어디로 문의하고, 어떤 언어로 답하나?	이름 없는 문의 양식뿐
떠나기: 표준 메서드인가? 월 단위 결제가 되나?	긴 의무 약정

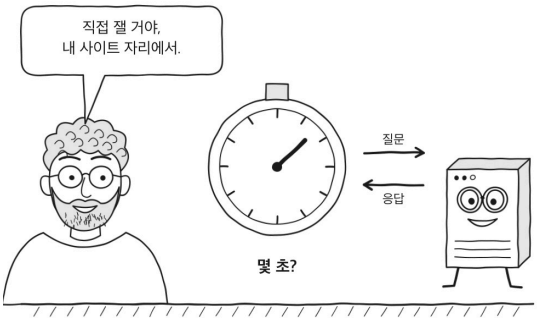
퍼센트가 숨기는 것

"가용성 99%"는 '항상'이나 다름없게 들려요. 30일 동안 얼마나 멈춰도 되는지 계산해 봅시다.

약속	멈춰도 되는 시간
99%	7시간 12분
99.9%	약 43분
99.99%	약 4분

그리고 정확히 무엇이 작동하는 것으로 치는지 물어보세요. 제공업체의 웹사이트인지, 내 네트워크의 내 엔드포인트인지요.

직접 해보기: 속도를 직접 재기



남이 잰 값은 남의 자리에서 잰 거예요. 여러분의 사이트가 살게 될 서버에서 요청을 다섯 번 보내 보세요.

```
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
```

숫자 다섯 개가 보일 거예요. 초 단위 응답 시간입니다. 가장 좋은 값만 보지 마세요. 손님은 가장 나쁜 값도 겪어요. 여기서는 매번 연결을 새로 여는데, 연결을 계속 유지하는 프로그램은 보통 더 빠릅니다. 속도뿐 아니라 블록의 신선도도 비교하세요(12장).

우리에게도 물어보세요

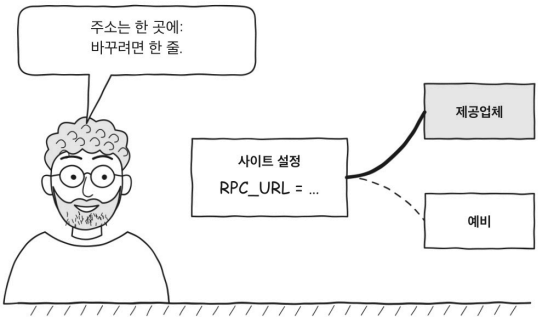
RpcNode에 관한 책이니 RpcNode도 점검해 볼게요. 2026년 11월 27일 기준입니다.

질문	RpcNode
네트워크	사이트 목록에 25개
무료 체험	Free 요금제와 키 없는 공개 노드
한도	Free: 초당 15건, 크레딧 1,500만. Starter: 초당 40건, 크레딧 5,000만
초과하면	초과 사용 추가 요금 없음: 요청이 거절됨(429 또는 402, 11장)
아카이브	유료 요금제에서
WebSocket과 gRPC	개인 엔드포인트로, 모든 네트워크는 아님
서버 위치	프랑크푸르트, 미국, 일본
보안	엔드포인트를 IP 주소와 사이트로 제한할 수 있음
프라이버시	제공업체는 요청을 볼 수 있음(3장)
가용성과 지원	rpcnode.dev 사이트의 현행 조건을 확인

함정: 가장 큰 숫자를 보고 고르기

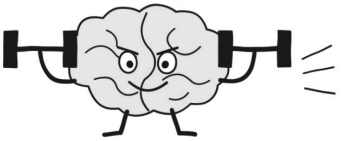
내가 쓰는 두 네트워크가 잘 안 돌아가면 천 개의 네트워크도 소용없어요. 필요한 메서드가 비싸거나 막혀 있으면 가장 싼 가격도 소용없고요. 무료나 가장 싼 요금제로 시작해서, 며칠 동안 진짜 내 요청을 돌려 보세요.

뿌리내리지 마세요



- 엔드포인트 주소는 코드가 아니라 설정 한 곳에 두세요.
- 가능하면 네트워크의 표준 메서드를 쓰세요.
- 예비 수단을 마련하고, 한 번은 시험 삼아 갈아타 보세요.

생각해 보기



머리 좀 굴려 볼까!

1. 주문 20건, 30초에 한 번 확인. 30일 동안 요청은 몇 건일까요?
2. 99.9%를 약속했는데, 엔드포인트가 한 달에 두 시간 멈췄어요. 약속은 지켜진 걸까요?
3. 엔드포인트 주소가 코드 14곳에 적혀 있어요. 무엇이 나쁠까요?

정답: 1) 1,728,000건. 2) 아니요. 허용되는 것은 약 43분이에요. 3) 바꿀 때 한 곳을 빠뜨리기 쉽고, 코드 속의 키는 새어 나가기 더 쉬워요.

핵심 정리

- 먼저 내게 필요한 것을 계산하고, 그다음 비교하세요.
- 약속은 내가 직접 잰 값과 내 요청으로 확인하세요.
- 떠날 수 있는 길을 남겨 두세요.

파트 3 끝. 아르티옴은 노드가 무엇인지, 어떻게 말을 거는지, 누구에게서 구하는지 알게 됐어요. 그런데 네트워크마다 다르잖아요! 다음: 파트 4, 14장. Bitcoin과 그 친척들.

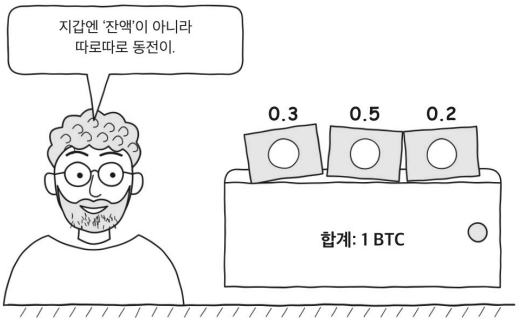
제14장. Bitcoin과 그 친척들

“그런데 잔액은 어디 있죠?”

손님이 비트코인으로 결제하고 싶어 해요. 아르티옴은 이제 경험자죠. Ethereum처럼 노드에게 주소 잔액을 물어보려는데, Bitcoin 노드에는 그렇게 간단한 질문이 없어요. 왜일까요?

알아보기: 계좌가 아니라 동전

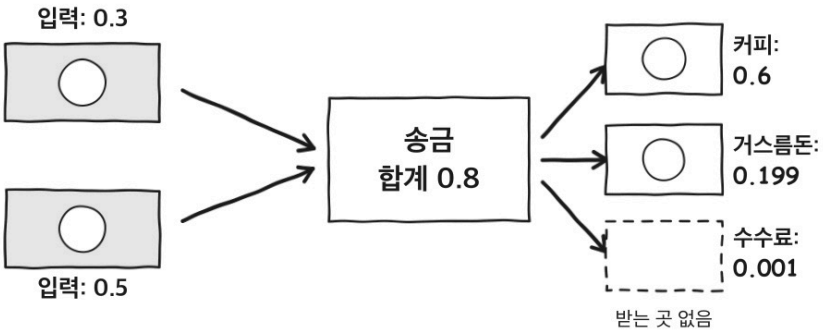
Ethereum에서는 주소에 은행 계좌처럼 잔액이 있어요. Bitcoin에는 없습니다. 대신 액면이 제각각인 지폐처럼 따로따로인 ‘돈 조각’이 있어요.



이런 조각 하나하나를 UTXO(미사용 출력)라고 해요. 잔액은 아직 쓰지 않은 내 조각의 합입니다. 지갑이나 인덱서가 계산해요(10장).

송금: 입력, 출력, 거스름돈

지폐는 살짝 찢어서 쓸 수 없어요. 통째로 내고 거스름돈을 받아야 하죠. 여기도 같아요. 송금은 옛 조각을 통째로 쓰고(입력), 새 조각을 만듭니다(출력).



- 출력 하나는 받는 사람에게.
- 또 하나는 나에게 돌아오는 거스름돈.
- 어느 출력에도 적히지 않은 몫은 채굴자가 가져가요. 이것이 수수료입니다. 입력에서 출력을 뺀 값이죠.

> 0.8 − 0.6 − 0.199 = 수수료 0.001 BTC

수수료 가격은 금액이 아니라 송금의 크기로 매겨요. 단위는 가상 바이트당 사토시입니다. 사토시는 1비트코인의 1억분의 1이에요. 입력 조각이 많을수록 송금이 커지고 비싸집니다.

사람 사이의 맨 첫 송금

2009년 1월 12일, 170번 블록에서 50 BTC 조각 하나가 10 BTC와 40 BTC 두 개로 바뀌었어요. 10을 받은 프로그래머 할 피니는 사토시가 시험 삼아 보내 준 것이라고 직접 이야기했습니다. 거스름돈 40 BTC는 보낸 사람에게 돌아갔어요. 수수료는 없었습니다.

블록체인에는 이름이 없어요. 피니의 이야기는 본인이 직접 밝혀서 알려진 것입니다(bitcointalk.org, 2013년).

이 송금은 지금도 조회할 수 있어요.

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

응답에서 `vin`(입력, 여기서는 하나)과 `vout`(출력, 여기서는 `10.0`과 `40.0` 두 개)를 찾아보세요.

직접 해보기: 이 동전은 이미 썼을까?

`gettxout` 메서드는 "이 조각이 아직 남아 있어?"라고 물어요. 조각은 송금 번호와 0부터 세는 출력 번호로 가리킵니다. 바로 그 10 BTC를 물어볼게요.

```
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`은 그런 미사용 조각이 없다는 뜻이에요. 오래전에 써 버렸어요. 이번에는 시작 블록 다음의 맨 첫 블록 보상을 물어봅시다.

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

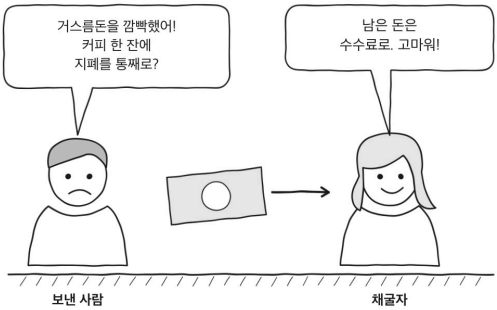
이번에는 `"value":50.00000000`과 거의 백만 개의 컨펌이 옵니다. 이 장을 쓰던 시점까지 이 50 BTC는 아무도 건드리지 않았어요.

Bitcoin에 관한 세 가지 사실 더

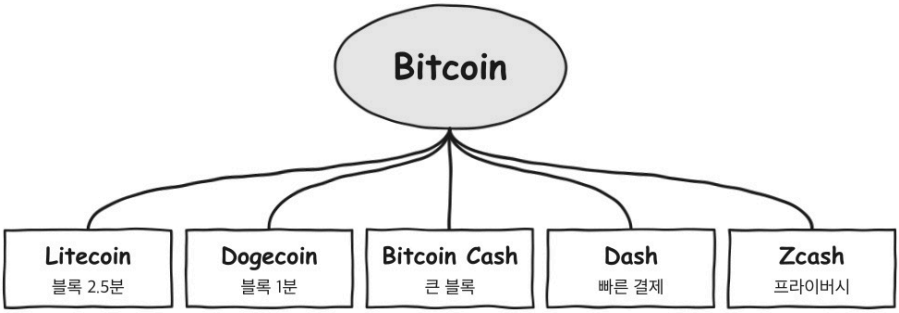
- 코인은 통틀어 2,100만 개를 넘지 않아요.
- 채굴자의 보상은 위에 100블록이 쌓이기 전에는 쓸 수 없어요.
- 큰 금액은 6컨펌, 약 1시간을 기다려요(8장).

함정: 잊어버린 거스름돈

보통의 지갑은 거스름돈을 알아서 넣어 줘요. 하지만 프로그램이 송금을 직접 조립하면서 거스름돈 출력을 빠뜨리면, 남은 돈 전부가 수수료가 됩니다.



친척들



구조는 친척, 하지만 네트워크와 코인은 각자

네트워크	블록 간격 (대략)	다른 점	주소의 시작
Bitcoin	10분	집안의 맏이	`1`, `3`, `bc1`
Litecoin	2.5분	다른 채굴 알고리즘 Scrypt	`L`, `M`, `ltc1`
Dogecoin	1분	발행량 상한 없음. Litecoin과 함께 채굴 가능	`D`
Bitcoin Cash	10분	훨씬 큰 블록	`bitcoincash:q…`
Dash	2.5분	빠른 결제(InstantSend)와 블록 교체 방지 (ChainLocks)	`X`
Zcash	75초	송금을 숨길 수 있음	공개 `t1`, 비공개 `zs`, `u1`

구조가 비슷해서 질문도 거의 같아요. 다섯 네트워크의 체인 높이를 한꺼번에 물어볼게요.

```
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
```

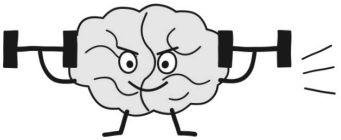
```
bitcoin: {"jsonrpc":"2.0","result":977717,"id":1}
litecoin: {"result":3219941,"error":null,"id":1}
doge: {"result":6474162,"error":null,"id":1}
bch: {"result":979125,"error":null,"id":1}
dash: {"result":2578936,"error":null,"id":1}
```

두 번째 함정: “error가 있으니 오류”

응답을 보세요. 네 네트워크에는 `error` 필드가 늘 붙어 있어요. 값이 `null`이면 “다 괜찮다”는 뜻입니다. 11장의 조언대로, 필드가 있는지가 아니라 안에 무엇이 들었는지를 확인하세요.

그리고 네트워크를 헷갈리지 마세요. 친척끼리도 서로의 코인은 받지 않아요. Bitcoin Cash의 옛 주소 형식은 Bitcoin과 똑같이 생겨서, `bitcoincash:`라는 접두어가 붙은 새 형식을 도입했습니다.

생각해 보기



1. 지갑에 0.3, 0.5, 0.2 BTC 조각이 있어요. 0.4를 내야 하고 수수료는 0.001이에요. 조각 하나로 충분할까요? 거스름돈은 얼마일까요?
2. 작은 조각 백 개를 모아 만든 1 BTC를 보내는 것이 통짜 1 BTC보다 왜 비쌀까요?
3. 입력은 1.0 BTC, 출력은 0.4 BTC 하나뿐이에요. 0.6은 어디로 갔을까요?

정답: 1) 네, 0.5 조각이면 돼요. 거스름돈은 0.099. 2) 입력이 많을수록 송금의 크기가 커져요. 3) 수수료예요. 아마 거스름돈을 빠뜨렸을 거예요.

핵심 정리

- Bitcoin에는 계좌 잔액이 없고, 쓰지 않은 조각(UTXO)이 있어요.
- 수수료는 입력에서 출력을 뺀 값이에요. 거스름돈을 잊지 마세요.
- 친척들은 구조가 비슷하지만 서로 다른 네트워크예요.

다음: 15장. Ethereum: 계정, 가스, 이벤트.

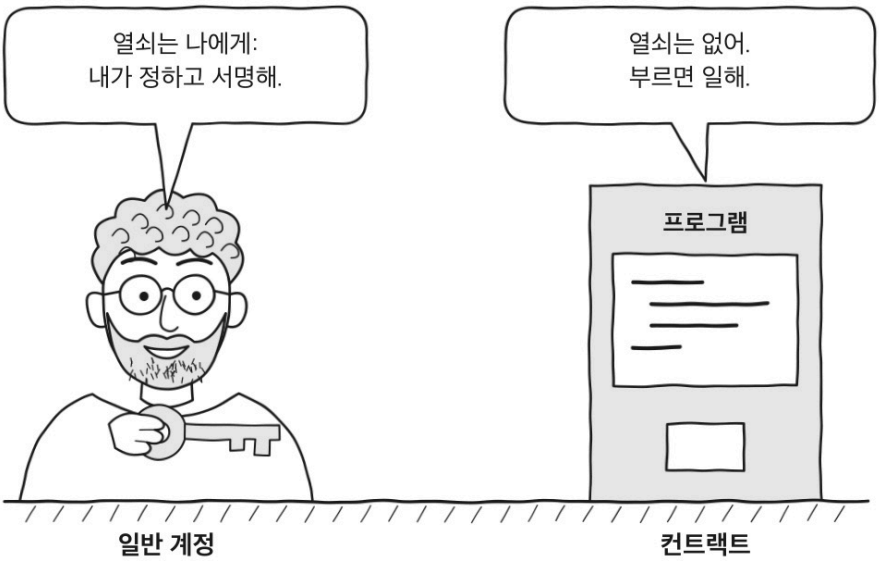
제15장. Ethereum: 계정, 가스, 이벤트

“왜 달러를 보내는 게 더 비싸죠?”

아르티옴이 공급업체에게 ETH를 조금 보내고, 그다음 USDT를 보냈어요. 금액은 비슷한데 USDT에는 ‘가스’가 두 배도 넘게 들었어요. 가스가 뭐고, 왜 그럴까요?

알아보기: 두 종류의 계정

Ethereum에서는 주소에 은행처럼 잔액이 있어요. Bitcoin의 조각(14장)과는 다릅니다. 그리고 계정에는 두 종류가 있어요.



	일반 계정	컨트랙트
누가 다루나	키의 주인	자기 프로그램
스스로 송금을 시작할 수 있나	예	아니요, 호출될 때만
코드가 있나	보통 없음	있음

프로그램은 EVM, 곧 Ethereum 가상 머신이 실행해요. 노드마다 들어 있는 게임기 같은 것이라, 같은 프로그램은 어디서나 같은 결과를 냅니다.

직접 해보기: 프로그램은 어디에 있을까

`eth_getCode` 메서드는 주소에 있는 코드를 보여 줘요. USDT 컨트랙트를 물어보고, 응답의 앞부분만 남겨 볼게요.

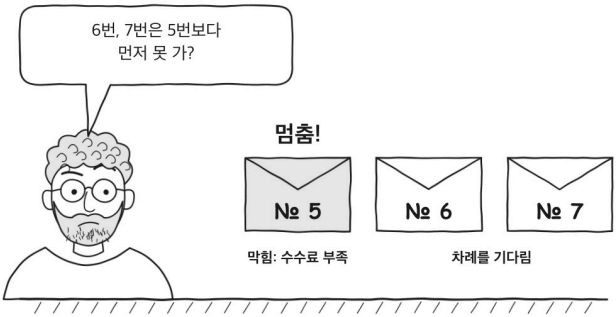
```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getCode","params":
  ["0xdac17f958d2ee523a2206206994597c13d831ec7","latest"]}' | head -c 80; echo
```

긴 글자 사슬의 첫머리가 보일 거예요. 이것이 바로 프로그램이고, 1만 1천 바이트가 넘습니다. 주소를 `0x000000000000000000000000000000000000dEaD`로 바꾸면 `"0x"`가 와요. 코드가 없다는 뜻입니다.

2025년부터는 일반 계정도 자기 동작을 프로그램에 맡길 수 있어요(Pectra 업그레이드). 이때 응답은 `0xef0100`으로 시작합니다. 그러니 "코드가 있다"가 늘 "컨트랙트다"를 뜻하지는 않아요.

순번: nonce

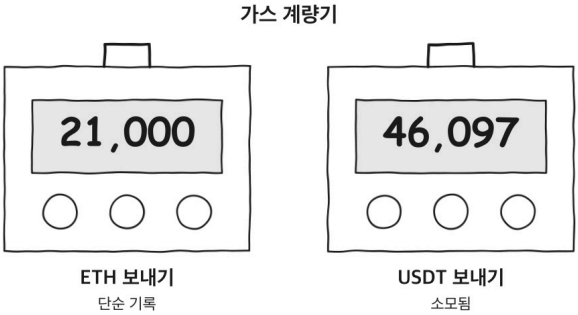
계정의 모든 트랜잭션에는 번호가 붙어요. 0, 1, 2 하는 식이죠. 이것을 nonce라고 합니다. 네트워크는 반드시 순서대로 실행하고, 같은 번호는 두 번 받지 않아요.



수수료가 적은 송금이 막히면 뒤의 것들도 기다려요. 지갑은 '가속' 기능을 제공하는데, 같은 번호에 더 큰 수수료를 붙인 새 송금을 보내는 것입니다.

가스: 작업량 계량기

가스는 트랜잭션에 필요한 작업량을 재요. 단순한 "ETH 보내기"는 현재 21,000입니다. 토큰 송금은 프로그램을 실행하는 것이라 가스가 더 필요해요.



수수료 = 사용한 가스 × (기본 가격 + 팁)

- 기본 가격은 네트워크가 직접 정해요. 이 돈은 소각됩니다.
- 팁은 검증자가 받아요. 팁이 높을수록 기꺼이 태워 줍니다.
- 가스 가격은 gwei로 세요. ETH의 10억분의 1입니다.

블록에 담을 수 있는 가스에는 한도가 있어요. 블록이 절반 넘게 차면 기본 가격이 오르고, 절반보다 덜 차면 내려갑니다. 예를 들어 26,502,250번 블록은 가스 6,000만을 담을 수 있었고 93% 차 있었어요.

지갑은 가스 한도를 정해요. 실제로 쓴 만큼만 빠져나갑니다. 하지만 한도가 모자라면 송금은 실행되지 않고, 이미 쓴 가스는 돌려받지 못해요.

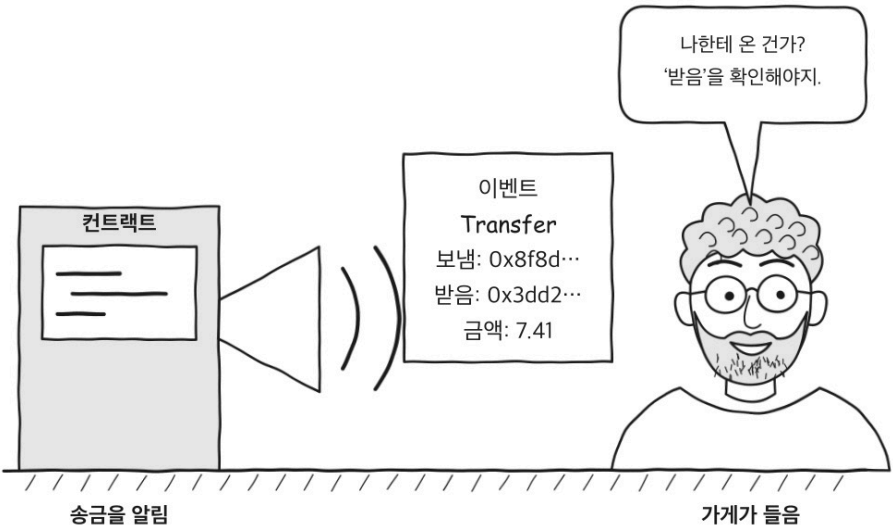
진짜 USDT 송금

바로 그 블록에 있는 7.41 USDT 송금이에요. 보낸 사람은 그때까지 이미 423,464건의 트랜잭션을 보냈어요.

항목	값
사용한 가스	46,097
기본 가격	0.058 gwei
팁	2 gwei
수수료	약 0.000095 ETH
그중 소각	약 0.000003 ETH

이벤트: 컨트랙트가 송금을 알리는 법

토큰 잔액은 컨트랙트 안에 있어요. 송금을 세상에 알리려고 컨트랙트는 이벤트(log)를 발행합니다. 토큰의 경우 `Transfer` 라고 해요. 누가, 누구에게, 얼마를 보냈는지 담깁니다.



이벤트는 7장의 트랜잭션 영수증 안에 있어요. 이 송금의 영수증을 달라고 해 볼게요.

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

응답에서 `logs`를 찾아보세요.

필드	앞과 끝	뜻
`address`	`0xdac1…1ec7`	알린 쪽: USDT 컨트랙트
`topics[0]`	`0xddf2…b3ef`	이벤트 종류: Transfer
`topics[1]`	`…8f8d…884e`	보낸 쪽
`topics[2]`	`…3dd2…a9c0`	받는 쪽
`data`	`…711150`	7,410,000, 즉 7.41 USDT

그리고 `eth_getLogs` 메서드는 이벤트를 검색할 수 있어요. 가장 최신 블록의 USDT 송금을 세어 봅시다.

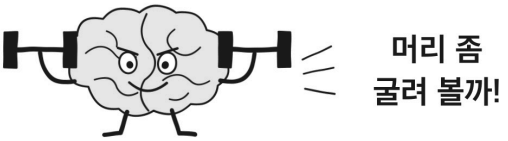
```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getLogs","params":[{
  "address":"0xdac17f958d2ee523a2206206994597c13d831ec7",
  "topics":["0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef"],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

블록 하나에 든 송금 수가 나와요. 지난 블록에 대해서는 공개 노드가 대답하지 않습니다. 그건 이미 아카이브니까요(10장).

함정: "이벤트만 듣기"

- ETH 자체의 송금은 Transfer 이벤트를 만들지 않아요. 이벤트만 기다리는 가게는 이런 결제를 놓칩니다. 앞으로의 네트워크 업그레이드로 바뀔 수는 있어요.
- ETH는 컨트랙트에서 올 수도 있어요. 예를 들어 어떤 지갑에서요. 이때는 트랜잭션 자체에 여러분의 주소가 보이지 않습니다.
- 이벤트는 어떤 컨트랙트든 발행할 수 있어요. `address`를 확인하세요. 진짜 토큰인지요(9장).

생각해 보기



1. ETH 송금, 가스 21,000, 팁을 포함한 가격 2 gwei. 수수료는 gwei로, 그리고 ETH로 얼마일까요?
2. 블록 몇 개가 연달아 30%만 찼어요. 기본 가격은 어떻게 될까요?
3. 가게는 USDT 컨트랙트의 Transfer 이벤트만 지켜봐요. 손님이 ETH로 결제했어요. 가게가 알 수 있을까요?

정답: 1) 42,000 gwei, 즉 0.000042 ETH. 2) 내려가요. 3) 아니요.

핵심 정리

- 계정은 키 아니면 프로그램이 다뤄요.
- 수수료는 가스에 가격(기본 가격과 팁)을 곱한 것이에요.
- 토큰 송금은 이벤트로 알 수 있고, ETH는 따로 잡아야 해요.

다음: 16장. Ethereum에는 친척이 많아요. 다른 EVM 네트워크들과 '2층'입니다.

제16장. Ethereum의 친척들과 '2층'

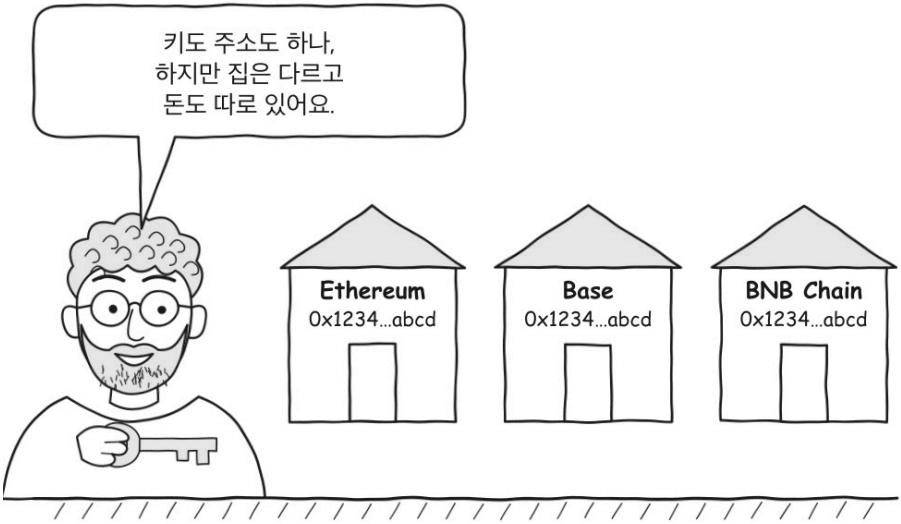
"Base로 보냈어요"

손님이 이렇게 써요. "Base에서 USDC를 보냈어요. 주소는 같던데요." 아르티옴은 어리둥절해요. 다른 토큰인가? 다른 네트워크인가? 그런데 주소는 왜 같지?

알아보기: 같은 게임기, 다른 집

15장의 EVM은 여러 네트워크가 가져다 썼어요. 그래서 이들은 주소가 같고, `eth_…` 메서드도 같고, 컨트랙트 프로그램 하나가 어디서나 돌아갈 수 있습니다. 이런 네트워크를 EVM 호환 네트워크라고 해요.

하지만 장부는 서로 달라요. 블록도, 잔액도, 수수료도 각자의 것입니다.



네트워크를 헷갈리지 않도록 저마다 번호가 있어요. 바로 chain ID입니다. 이 번호는 서명에 들어가기 때문에, 한 네트워크용으로 서명한 송금을 다른 네트워크는 받지 않아요.

직접 해보기: 여섯 네트워크에게 번호 묻기

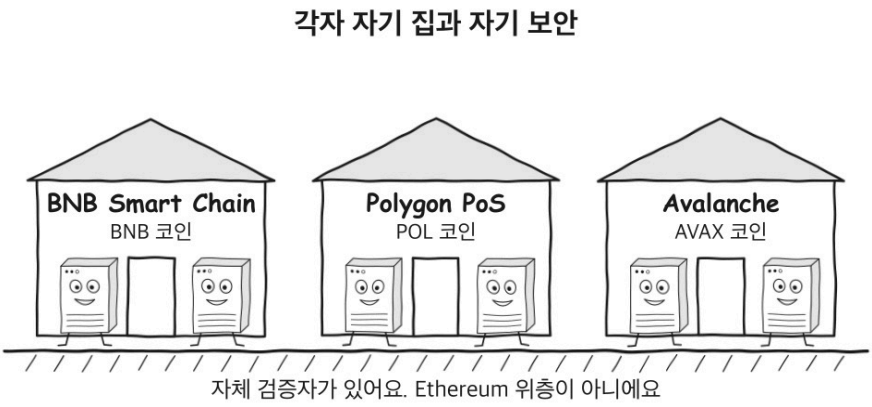
```
for net in ethereum base arb bsc mantle monad; do
  echo -n "$net: "
  curl -s https://public-$net-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_chainId","params":[]}'
  echo; sleep 1
done
```

16진수 숫자가 옵니다. 보통 숫자로 바꾸고, 확인 당시 블록이 얼마나 자주 나왔는지를 함께 적으면 이렇습니다.

네트워크	Chain ID	블록 간격(대략)	수수료 코인
Ethereum	1	12초	ETH
Base	8453	0.2초	ETH
Arbitrum One	42161	0.3초	ETH
BNB Smart Chain	56	0.45초	BNB
Mantle	5000	2초	MNT
Monad	143	0.3초	MON

첫 번째 부류의 친척: 독립 네트워크

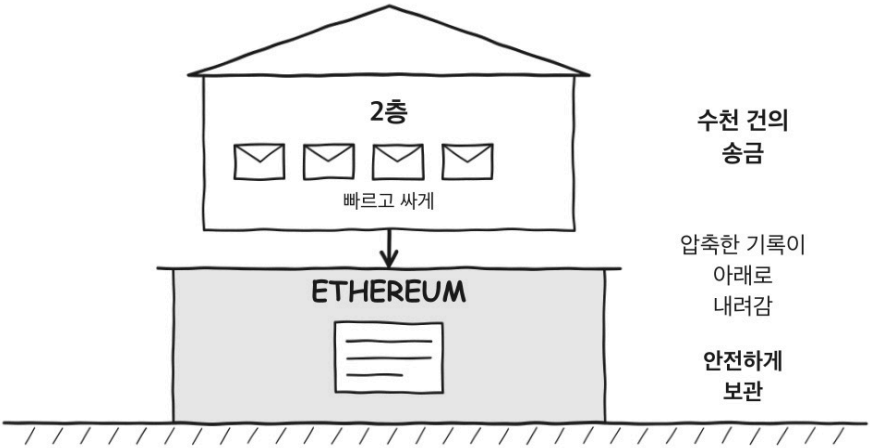
이들은 자기 검증자가 있고, 블록을 스스로 합의해요.



다만 Polygon PoS는 주기적으로 자기 블록의 표식을 Ethereum에 남겨요. 2025년 11월에 출범한 Monad도 독립 네트워크에 속합니다.

두 번째 부류의 친척: 2층(L2)

L2는 송금을 자기 안에서 빠르고 싸게 처리하고, 압축한 기록을 Ethereum으로 보내요. 가장 흔한 종류를 롤업(rollup)이라고 합니다. '돌돌 말아 올린다'는 뜻이에요.



- 송금 순서는 보통 한 운영자, 시퀀서가 정해요. 그래서 그렇게 빠릅니다. 그리고 그래서 이 지점에서는 그를 믿어야 해요.
- 옵티미스틱 롤업(Arbitrum One, OP Mainnet)은 기록이 맞다고 치되, 이의를 제기할 시간을 줘요.
- ZK 롤업(예를 들어 Mantle)은 수학적 증명을 붙여요.
- Base는 2026년에 자체 코드로 갈아탔고, 여러 종류의 증명을 결합해 써요.

검증 시간 때문에, 공식 브리지로 Ethereum에 돈을 되가져오는 것은 빠르지 않아요.

출발 네트워크	Ethereum으로 출금하는 데 걸리는 시간
Arbitrum One	최소 7일
OP Mainnet	7일
Base	5일, 경우에 따라 1일

2026년 11월 27일 기준 각 네트워크 문서에 따른 값이에요. 이 기간은 바뀝니다. OP Mainnet은 2027년 1월에 개편이 예정돼 있어요.

"빨리 보였다"가 "최종 확정"은 아니다

8장의 `latest`, `safe`, `finalized` 기억나죠? Base 노드는 이렇게 대답했어요.

무엇을 물었나	최신 블록과의 차이
`latest`	—
`safe`	약 17초
`finalized`	약 19분

블록은 1초도 안 돼 나타나지만, 최종 확정은 그 블록에 관한 기록이 Ethereum에서 최종 확정될 때 이루어져요.

2층의 수수료

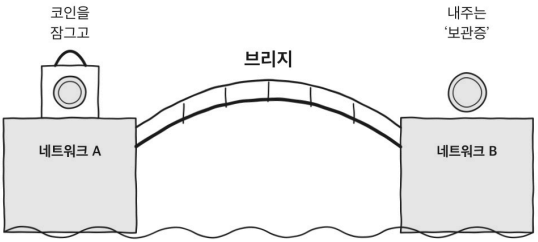
주로 두 부분으로 이루어져요. L2에서의 작업 값과, Ethereum에 기록하는 값입니다. 자잘한 추가분이 붙기도 해요. Base의 영수증에서 두 번째 부분은 `l1Fee` 필드에 있습니다.

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

여기서는 Base에서의 작업 값으로 약 0.0000035 ETH, Ethereum 기록 값으로 약 0.0000000006 ETH를 받았어요. 늘 이렇게 싼 것은 아닙니다.

브리지

네트워크에서 네트워크로 '이사'할 때는 브리지를 써요. 흔히 한 네트워크에서 코인을 잠그고, 다른 네트워크에서 대체물을 발행합니다.



보관증 값은 브리지가 믿을 만한 만큼

그래서 USDC와 USDC.e는 컨트랙트 주소가 다른, 서로 다른 토큰이에요. 앞의 것은 Circle 회사가 직접 발행하고, 뒤의 것은 브리지를 거쳐 생깁니다.

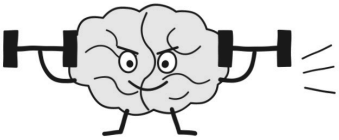
함정: 다른 네트워크로 보냈다면

6장에서 가게는 이런 결제를 보지 못한다고 했어요. 맞는 말이에요. 그런데 돈은 사라진 걸까요?

- 그 주소의 키가 나에게 있다면 보통은 돈을 꺼낼 수 있어요. 지갑을 그 네트워크로 바꾸세요. 다시 보내려면 수수료로 그 네트워크의 코인이 필요합니다.
- 거래소의 주소라면 나에게는 키가 없어요. 거래소가 그 네트워크를 지원하지 않으면, 돈은 아마 돌려받지 못할 거예요.
- 컨트랙트 지갑이라면 다른 네트워크에는 존재하지 않을 수 있어요. 전문가의 도움이 필요합니다.

원칙: 송금을 '꺼내 주겠다'고 약속하는 사람에게도 시드 문구는 알려 주지 마세요(6장).

생각해 보기



머리 좀 굴려 볼까!

1. 가게는 Ethereum의 USDT를 기다렸는데, BNB Smart Chain에서 같은 주소로 받았어요. 키는 아르티옴에게 있어요. 어떻게 해야 할까요?
2. Base의 블록은 0.2초 만에 나타났어요. 큰 금액의 결제가 최종 확정됐다고 봐도 될까요?
3. 손님이 USDC 대신 USDC.e를 보냈어요. 같은 것일까요?

정답: 1) 지갑을 BNB Smart Chain으로 바꿔요. 다시 보내려면 BNB가 필요해요. 2) 아니요, finalized를 기다리세요. 3) 아니요, 다른 토큰이에요. 받을지는 가게가 정해요.

핵심 정리

- EVM 네트워크는 비슷하지만 서로 다른 장부예요. chain ID를 확인하세요.
- 독립 네트워크도 있고, Ethereum 위의 2층도 있어요.
- 블록이 빠르다고 최종 확정은 아니에요.

다음: 17장. TRON: 가스 대신 에너지, 그리고 스테이블코인이 즐겨 다니는 길.

제17장. TRON: 가스 대신 에너지, 스테이블코인이 즐겨 다니는 길

“TRC-20으로 돼요?”

손님이 물어요. “USDT TRC-20으로 받으세요?” 아르티옴은 이번 주에만 벌써 세 번째 듣는 말이에요. TRC-20이 뭐길래 다들 그걸 찾을까요?

알아보기: 독립된 네트워크

TRON은 자기 코인 TRX를 가진 별도의 블록체인이에요. 가장 작은 단위는 sun이라고 해요. 1 TRX는 100만 sun입니다.

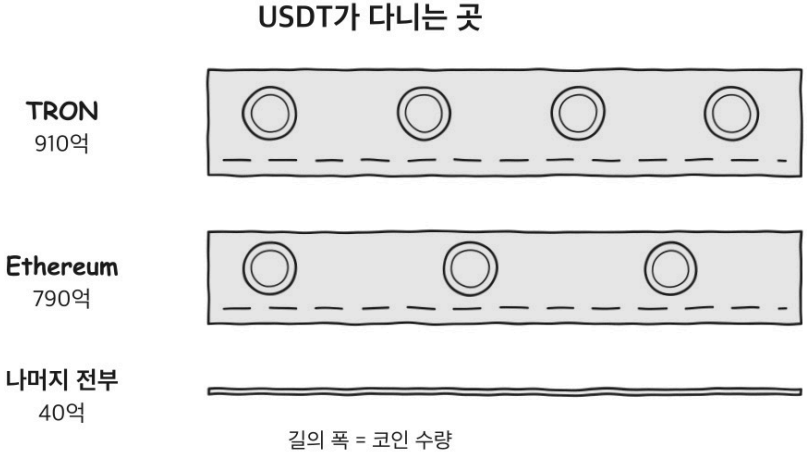
TRON의 구조는 여러모로 Ethereum과 비슷해요. 계정의 잔액, 스마트 컨트랙트, 이벤트가 있습니다. 프로그램은 15장의 EVM과 가까운 친척인 자체 가상 머신이 실행해요.

TRC-20은 TRON에서 토큰을 만드는 규칙으로, Ethereum의 ERC-20 표준의 형제예요. “TRC-20 USDT”는 TRON 네트워크의 USDT 토큰이라는 뜻입니다.

블록은 3초마다 나와요. 선출된 대표 27명이 씁니다(8장).

스테이블코인이 즐겨 다니는 길

USDT는 여러 네트워크에 살지만, 아주 고르지 않게 퍼져 있어요. 이 코인을 발행하는 Tether 회사의 숫자를 2026년 11월 27일에 확인한 것입니다.



전체 USDT의 절반 이상이 TRON에 있어요. 그래서 손님들이 이 네트워크를 그렇게 자주 고릅니다.

출처: app.tether.to/transparency.json. 발행된 코인에서 아직 유통되지 않은 재고를 뺀 값이에요.

주소는 T로 시작해요

TRON 주소의 속은 Ethereum처럼 20바이트이고, 앞에 `41`이 붙어요. 사람에게 보여 줄 때는 다르게, `T`로 시작하게 적습니다.

사람이 보는 모습	속의 모습 (USDT 컨트랙트 주소)
`TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t`	`41a614f803b6fd780986a42c78ec9c7f77e6ded13c`

`T`로 적은 형태에는 체크섬이 숨어 있어서 오타를 알아챌 수 있어요. 진짜 주소를 노드에게 물어보고, 마지막 글자를 바꿔서 다시 물어볼게요.

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

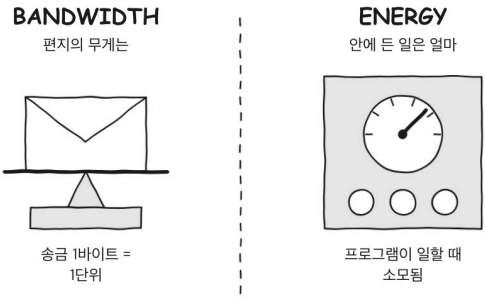
curl https://public-tron-mainnet.rpcnode.dev/wallet/validateaddress \
  -d '{"address":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6u","visible":true}'
```

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

`"visible":true`는 “주소를 T로 시작하는 형태로 적을게”라는 뜻이에요.

가스 대신 두 가지 요금

7장에서 TRON이 송금의 ‘무게’를 어떻게 재는지 알려 드리겠다고 했죠. TRON은 두 가지 척도로 잽니다.



- Bandwidth(대역폭)는 자리 값이에요. 소비량은 대략 송금의 바이트 크기와 같아요. 모든 송금이 냅니다.

- Energy(에너지)는 프로그램의 작업 값이에요. 컨트랙트를 호출할 때 필요하니, USDT를 보낼 때도 필요합니다.

TRX를 한 번이라도 받은 주소에는 선물이 있어요. 하루에 bandwidth 600단위입니다. 쓴 만큼은 24시간에 걸쳐 조금씩 다시 차요. 무료 에너지는 없습니다.

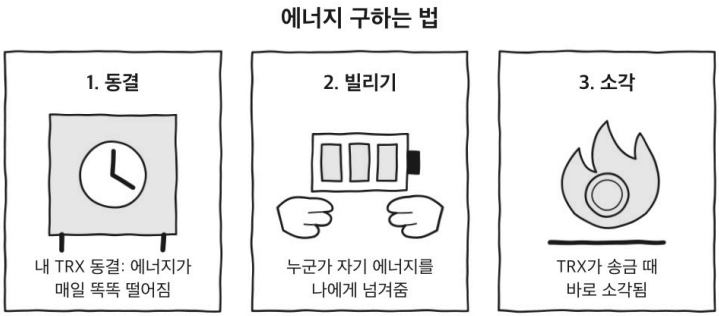
무언가 모자라면 네트워크가 TRX를 가져다 태워요. 얼마에? 노드에게 직접 물어봅시다.

```
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
```

즉 bandwidth 한 단위는 1000 sun, 에너지 한 단위는 100 sun이에요. 이 가격은 대표들의 투표로 바뀝니다.

에너지는 어디서 구할까

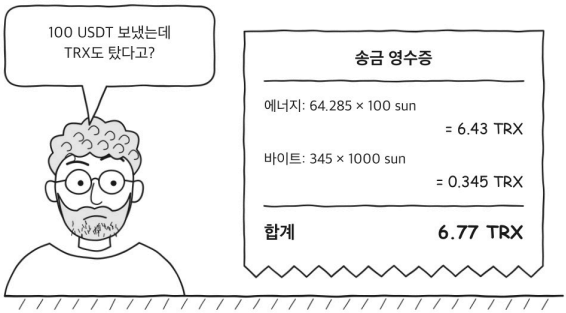


- TRX 동결하기(영어로 staking). 코인은 계속 내 것이고, 주소는 매일 전체 에너지 가운데 한 몫을 받아요. 코인을 되찾으려면 동결을 해제한 뒤 14일을 기다립니다.
- 빌리기: TRX를 동결한 주인이 자기 에너지를 다른 주소에 넘겨줘요.
- 아무것도 안 하기: 모자란 몫은 TRX가 소각되어 치러져요.

얼마나 동결해야 할까요? 다른 모든 사람이 얼마나 동결했는지에 달려 있어요. 확인한 날 기준으로, 하루에 USDT 한 번 보내려면 동결한 TRX 약 6600개면 충분했습니다.

진짜 송금의 영수증

88,352,467번 블록에 있는 100 USDT 송금이에요. 보낸 사람에게는 에너지도, 남은 무료 bandwidth도 없었어요.



영수증은 `gettransactioninfobyid` 메서드로 가져와요. 공개 노드에서는 막혀 있고 개인 엔드포인트가 필요해서, 아래는 대강의 모습입니다.

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

응답의 필드	값	뜻
`energy_usage_total`	64,285	사용한 에너지 총량
`energy_penalty_total`	49,635	그중 인기 할증분
`energy_fee`	6,428,500	에너지 값으로 소각된 sun
`net_fee`	345,000	바이트 값으로 소각된 sun
`fee`	6,773,500	합계, 즉 6.7735 TRX
`result`	`SUCCESS`	프로그램이 성공적으로 실행됨

할증이 뭘까요? 한 컨트랙트가 너무 많이 쓰이면, 네트워크가 그 호출을 더 비싸게 만들어요. 그날 USDT에는 할증이 없을 때보다 거의 4.4배의 에너지가 들었습니다.

또 하나. 받는 사람에게 그전까지 USDT가 없었다면 에너지가 두 배, 130,285가 필요해요. 컨트랙트가 그를 위한 새 줄을 만들어야 하거든요.

최신 블록과 확정 블록

TRON 노드에는 주소가 두 벌 있어요. `/wallet/…`은 가장 최신 데이터로 답하고, `/walletsolidity/…`는 최종 확정된 데이터로만 답합니다. 블록은 대표 27명 중 19명이 그 높이 또는 그 위에 자기 블록을 냈을 때 확정돼요. 보통 약 1분입니다.

확인한 날, 최신 블록은 88,352,763번이었고 확정 블록은 88,352,746번이었어요.

실험용으로는 테스트넷 Nile도 있고, 전용 공개 노드가 있어요. `public-tron-nile.rpcnode.dev` 입니다.

함정: “블록에 있으니 들어왔겠지”

마지막 확정 블록에 있는 USDT 직접 송금을 살펴봅시다.

```
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('USDT 송금:', len(rows))
print('실패:', sum(r[1] != 'SUCCESS' for r in rows))
for r in rows[:3]:
    print(*r)
EOF
```

숫자는 여러분마다 다르게 나올 거예요. 88,352,467번 블록에서는 이런 송금 41건 중 두 건이 실패했어요. 그중 하나는 100,000 USDT짜리였습니다.

```
fbfec9fa OUT_OF_ENERGY 100000.0
```

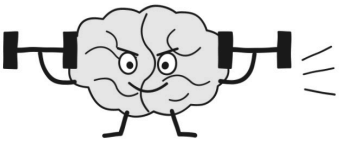
`OUT_OF_ENERGY` 는 에너지가 모자라 프로그램이 멈췄고, USDT는 아무 데도 가지 않았다는 뜻이에요. 그런데 블록에는 기록이 남아 있고, 화면 캡처에서는 금액이 그럴듯해 보입니다.

에너지를 얼마까지 쓸 수 있는지는 보내는 사람이 정해요. `fee_limit` 필드이고, sun 단위의 상한입니다. 상한이 모자라면 송금은 이루어지지 않고, 소각된 것은 돌려받지 못해요.

가게가 기억할 것

- 확정 블록을 기다리고, 영수증의 `result` 를 확인하세요.
- 영수증에서 `TR7NHq…Lj6t` 컨트랙트의 `Transfer` 이벤트를 찾으세요. 그러면 다른 프로그램을 거친 송금도 보여요. USDT는 소수점 아래 여섯 자리예요.
- 내 주소에서 USDT를 다시 보내려면 그 주소에 TRX나 에너지가 있어야 해요. 주문마다 주소를 두면 편하지만, 주소 백 개에서 돈을 모으려면 수수료 내는 송금이 백 번 필요합니다.
- 새 주소로 TRX를 맨 처음 보낼 때는, 보내는 사람에게 최소 1 TRX가 더 들어요.

생각해 보기



머리 좀 굴려 볼까!

1. 어떤 주소에 무료 bandwidth 600과 넉넉한 에너지가 있어요. 345짜리 송금을 연달아 두 번 보내요. 두 번째는 어떻게 될까요?
2. 에너지가 없고, 받는 사람은 USDT를 가진 적이 없어요. 100 sun짜리 단위가 130,285개 필요해요. 에너지 값으로 TRX가 얼마나 소각될까요?
3. 손님이 해시를 보냈어요. 트랜잭션은 블록에 있는데 `OUT_OF_ENERGY` 라고 돼 있어요. 커피를 보낼까요?

정답: 1) 남은 무료분은 255라 345에는 모자라요. 345 전부에 대해 0.345 TRX가 소각돼요. 나눠서 합치지는 않아요. 2) 13.0285 TRX. 3) 아니요, 송금은 없었어요.

핵심 정리

- TRON에는 두 가지 척도가 있어요. 바이트에는 bandwidth, 프로그램 작업에는 energy.
- 에너지가 없으면 TRX가 소각돼요. USDT는 단순 송금보다 훨씬 많이 들어요.
- 결제란 확정 블록에 담긴 진짜 USDT의 성공한 송금이에요.

다음: 18장. Solana: 모든 것이 계정에 나뉘어 있어요.

제18장. Solana: 모든 것이 계정에 나뉘어 있다

"내 USDC 어디 갔지?"

손님이 USDC로 결제했어요. 아르티옴이 노드에게 자기 지갑 잔액을 물으니 SOL 부스러기만 보여요. USDC는 없고요. 속은 걸까요?

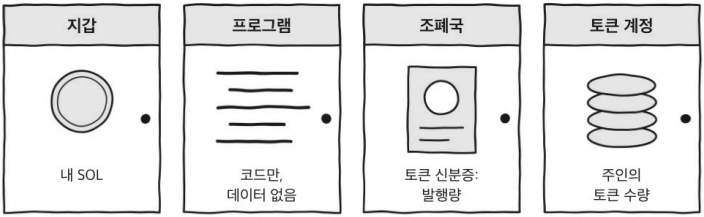
알아보기: 계정으로 된 네트워크

Solana는 SOL 코인을 가진 독립 네트워크예요. 가장 작은 단위는 램포트(lamport)이고, 1 SOL은 10억 램포트입니다.

Solana 주소는 32바이트를 글자와 숫자로 적은 거예요. 특별한 시작 글자도 없고(6장), 안에 체크섬도 없습니다.

네트워크가 저장하는 모든 것은 계정(accounts)에 들어 있어요. 여기서 계정은 돈만 뜻하는 게 아니라, 주소가 붙은 모든 칸을 말합니다. Ethereum에서는 컨트랙트가 데이터를 자기 안에 보관해요. Solana에서는 프로그램 따로, 데이터 따로입니다.

계정 넷, 모두 달라요



영어로: wallet, program, mint account, token account

모든 계정에는 주인 프로그램이 있어요(`owner` 필드). 오직 그 프로그램만 계정의 데이터를 바꿀 수 있습니다.

직접 해보기: 토큰의 신분증

모든 토큰에는 신분증 계정이 있고, 이것을 mint(민트)라고 불러요. USDC의 신분증을 물어봅시다.

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

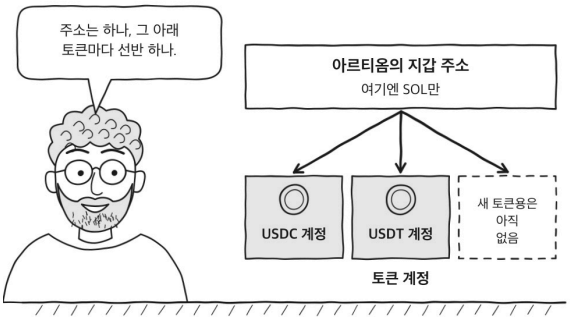
응답의 핵심은 이렇습니다.

필드	값	뜻
`owner`	`Tokenkeg…Q5DA`	토큰 프로그램이 계정을 관리함
`supply`	`8513455604386747`	발행량, 최소 단위로
`decimals`	6	즉 약 85억 USDC
`mintAuthority`	`BJE5…5ruG`	새로 발행할 수 있는 쪽
`freezeAuthority`	`7dGb…Crar`	동결할 수 있는 쪽(9장)

USDC에는 자기 프로그램이 없어요. 공용 토큰 프로그램 하나가 수천 개의 토큰을 맡고, 토큰들은 신분증으로 구별됩니다. 이런 공용 프로그램은 두 가지예요. 보통의 Token(USDC와 USDT가 여기 있어요)과 추가 기능이 있는 더 새로운 Token-2022입니다.

토큰 계정: 따로 있는 선반

아르티옴에게 USDC가 얼마 있는지는 지갑에도, 신분증에도 적혀 있지 않아요. 토큰 계정에 적혀 있습니다. 토큰마다 따로 있어요.



기본 토큰 계정을 ATA(associated token account)라고 해요. 그 주소는 지갑 주소, 토큰 신분증, 토큰 프로그램, 이 세 가지로 계산됩니다. 그래서 손님에게는 지갑 주소를 알려 주고, 선반은 손님의 프로그램이 알아서 찾아요.

진짜 토큰 계정을 살펴봅시다. 먼저 토큰이 얼마나 있는지, 그다음 SOL이 얼마나 있는지 물어볼게요.

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

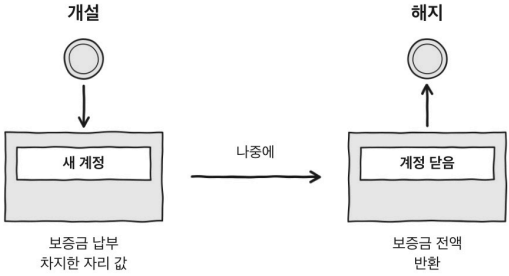
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getBalance","params":
  ["BnGihgNUu4X1JdBMNhbWnXUcpib8UQQC154isHTYwDNK"]}'
```

첫 번째 응답의 `"uiAmountString":"207637.336362"` 는 USDT예요. 두 번째의 `"value":2039280` 은 램포트이고, 다 해 봐야 0.00203928 SOL입니다. `getBalance` 메서드는 토큰을 보지 못해요. 아르티옴의 USDC가 '사라진' 곳이 바로 여기였어요.

그런데 이 선반의 주인은 USDT 신분증 주소 자체예요. 9장처럼, 사람들이 실수로 "USDT 주소로" USDT를 보낸 것 같아요.

자리 보증금

모든 계정은 모든 노드의 메모리를 차지해요. 그래서 계정에는 SOL 보증금이 들어 있어요 (영어로 rent, '임대료'라고 하지만 임대는 아니에요).



요금이 아니라 보증금: 토큰 계정당 약 0.0015 SOL

토큰 계정의 크기는 165바이트예요. 이만큼에는 보증금이 얼마나 필요할까요?

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

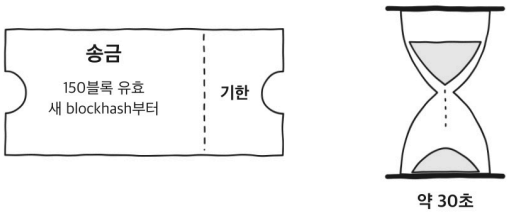
```
{"jsonrpc":"2.0","result":1488440,"id":1}
```

약 0.0015 SOL이에요. 2026년에는 보증금을 단계적으로 낮추고 있어서, 프로그램에 숫자를 적어 두지 않고 물어봅니다. 받는 사람에게 아직 그 토큰의 선반이 없으면, 보통 보내는 사람이 같은 송금 안에서 선반을 만들고 보증금도 넣어요.

수수료와 유효 기간

기본 수수료는 서명 하나당 5000램포트예요. 서명 하나짜리 보통 송금이면 0.000005 SOL입니다. 절반은 소각돼요. 여기에 급행료를 더 낼 수 있고, 그것은 검증자가 전부 가져갑니다. 송금이 실패해도 수수료는 받아요.

모든 송금에는 blockhash, 즉 최근 블록의 지문을 넣어요. 이것이 유효 기간도 정합니다.



블록에 못 들어가면 영영 못 들어가요

```
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"}]}'
```

첫 번째 응답에서는 `"lastValidBlockHeight":452330099`가 왔고, 1초 뒤 두 번째 응답의 높이는 452,329,960이었어요. 139블록이 남은 거죠. 기한이 지났는데 송금이 블록에 없으면, 이제는 통과되지 않아요. 그때는 새로 서명하면 됩니다. 예외는 특별한 예약 송금(durable nonce)으로, 이런 기한이 없어요.

2026년: 더 빨라지다

바뀐 것	이전	이후
슬롯, 즉 블록 하나의 차례 길이 (10월)	0.4초	0.2초
최종 확정, 새 투표 방식 Alpenglow (10월)	약 13초	약 0.15초
트랜잭션 최대 크기, 새 v1 형식 (9월)	1232바이트	4096바이트

속도를 직접 확인해 봅시다. 이 메서드는 1분 동안 지나간 슬롯 수를 돌려줘요.

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

저희는 60초에 `numSlots` = 277을 받았어요. 슬롯당 약 0.22초입니다. 슬롯은 비어 있을 때도 있어서, 슬롯 번호와 블록 높이는 서로 다른 숫자예요. 헷갈리지 마세요.

노드는 엄격함의 정도를 달리해 답할 수 있어요. `processed`는 블록을 봤지만 아직 건너뛸 수 있다는 뜻이고, `finalized`는 최종 확정입니다. 중간 단어 `confirmed`도 있는데, Alpenglow 이후로는 `finalized`와 같은 뜻이에요. 주문은 `finalized`를 보고 발송합니다.

함정: “트랜잭션이 안 읽혀요”

새 v1 형식 때문에 `getTransaction`과 `getBlock` 메서드에는 값이 1인 `maxSupportedTransactionVersion`을 넣어야 해요. 없으면 새 트랜잭션에서 오류가 납니다. `getTransaction` 메서드는 공개 노드에서 막혀 있어서, 아래는 개인 엔드포인트용 대강의 모습이에요.

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

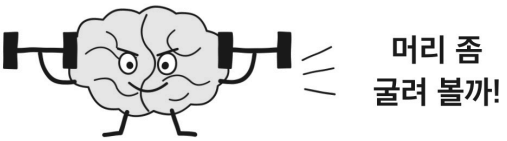
Solana에서는 ‘송금 서명’이 트랜잭션 해시 대신 송금의 번호 노릇을 해요.

가게가 확인할 것

- 트랜잭션이 성공했는지: `meta.err`가 `null`이어야 해요.
- 맞는 토큰 신분증을 가진 내 지갑의 토큰 계정이 맞는 금액만큼 늘었는지: `preTokenBalances`와 `postTokenBalances`를 비교하세요.
- 바로 이 주문의 결제인지. 이를 위해 송금에 reference, 즉 주문의 일회용 표식 주소를 넣어요.
- 이 서명을 이미 인정한 적은 없는지.

그리고 보통의 SOL은 더 조심하세요. 잘못된 주소로 보내도 송금은 실행됩니다. 이런 돈은 돌려줄 사람이 아마 없을 거예요.

생각해 보기



1. 지갑이 송금에 서명했는데, 네트워크는 2분 뒤에야 받았어요. 어떻게 될까요?
2. 가게가 USDC 결제를 잡으려고 자기 지갑의 `getBalance`를 물어요. 무엇이 잘못일까요?
3. 아르티옴이 아직 그 선반이 없는 공급업체에게 USDC를 보내요. 수수료 말고 무엇을 더 내야 할까요?

정답: 1) 기한이 지나 송금이 통과되지 않아요. 2) `getBalance`는 SOL만 보여 줘요. USDC는 토큰 계정에 있어요. 3) 새 토큰 계정의 보증금, 약 0.0015 SOL.

핵심 정리

- Solana에서는 모든 것이 계정이에요. 프로그램과 데이터는 따로 있어요.
- 토큰은 토큰 계정에 있어요. 토큰마다 하나씩, 보증금과 함께.
- 트랜잭션은 `maxSupportedTransactionVersion`을 붙여 읽고, `finalized`를 기다려요.

다음: 19장. TON: 전화 대신 편지.

제19장. TON: 전화 대신 편지

“보냈는데, 그다음은요?”

손님이 TON 네트워크에서 USDT를 보냈어요. 그런데 탐색기에는 기록이 하나가 아니라 긴 사슬로 보여요. 트랜잭션 다섯 개에, 낯선 주소들, 자잘한 거스름돈까지. 이건 결제 한 건일까요, 다섯 건일까요?

알아보기: 모두가 컨트랙트, 모두가 편지를 쓴다

TON(The Open Network)은 독립 네트워크예요. 코인 이름은 Gram(GRAM)입니다. 예전에는 Toncoin이라고 불렸고, 이 이름은 아직 자주 보여요. 1 Gram은 10억 나노그램입니다.

2026년 4월부터 블록은 약 0.4초마다 나오고, 기록은 약 1초 만에 최종 확정돼요.

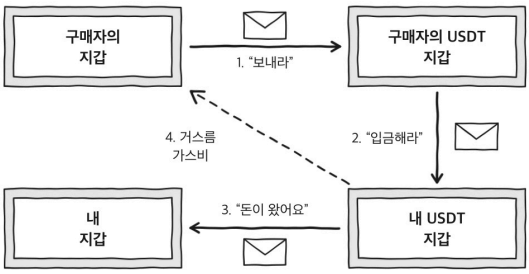
가장 큰 차이는 이거예요. 여기서는 모든 계정이 스마트 컨트랙트이고, 지갑도 마찬가지입니다. 컨트랙트끼리는 서로 직접 호출하지 않고 메시지를 보내요.

Ethereum에서는 송금 하나가 일으킨 모든 일이 트랜잭션 하나 안에서 벌어져요. 전부 되거나, 아예 안 되거나. 전화 통화와 비슷합니다. TON은 편지 주고받기예요. 편지를 보내고 수화기를 들고 기다리지 않아요. 받는 쪽은 편지 하나하나를 나중에, 별도의 트랜잭션으로 처리합니다. 이런 방식을 비동기라고 해요.

하나의 지시에서 뻗어 나온 편지 전체를 trace(자취)라고 불러요. 그 안의 트랜잭션 하나가 성공했다고 일 전체가 성공한 것은 아닙니다.

USDT는 어떻게 가나

TON의 토큰은 jetton(제톤)이라고 해요. 토큰마다 대표 컨트랙트(master)가 있고, 보유자마다 자기만의 jetton 지갑이 따로 있습니다. 18장의 토큰 계정과 비슷하지만, 여기서는 독립된 컨트랙트예요. USDT의 대표 컨트랙트는 `EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs` 입니다.



편지마다 별도의 트랜잭션으로 처리

지어낸 이야기가 아니라, 2026년 11월 27일로 넘어가는 밤에 실제로 이루어진 11.99 USDT 송금이에요.

누구의 트랜잭션	무엇을 받았나	무엇을 보냈나
보내는 사람의 지갑	서명된 지시	“보내라”와 비용 0.05 Gram
그의 USDT 지갑	“보내라”	“입금해라”
받는 사람의 USDT 지갑	“입금해라”	알림과 거스름돈
받는 사람의 지갑	알림과 1나노그램	없음
보내는 사람의 지갑	거스름돈 약 0.0494 Gram	없음

이 모든 것이 1초 안에 끝났어요. 비용은 넉넉하게 붙여 보내고, 남은 것은 돌아옵니다.

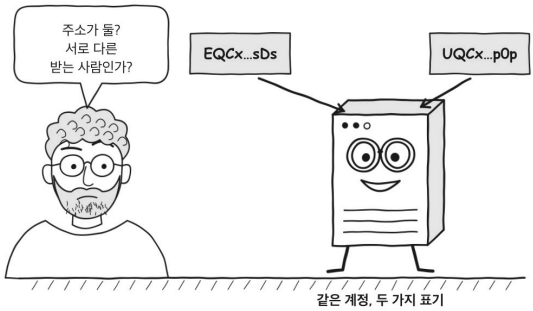
흥미로운 점 하나. 네 번째 트랜잭션에는 중단됨(`aborted`) 표시가 있어요. 받은 것이 1나노그램뿐이라 받는 사람 쪽에서 프로그램을 실행할 가스가 모자랐던 거예요. 하지만 USDT는 이미 세 번째 트랜잭션에서 입금됐습니다. 기록 하나만 보고는 어느 쪽으로도 판단할 수 없어요.

사슬 전체는 인덱서에 요청해서 볼 수 있어요. 공개 노드에서는 막혀 있어서, 아래는 개인 엔드포인트용 대강의 모습입니다.

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

계정은 하나, 주소는 둘

계정의 진짜 이름은 ‘원시(raw)’ 형태예요. 네트워크 구역 번호와 64자리입니다. 사람에게는 체크섬이 든 48자의 다른 표기를 보여 줘요. 그리고 계정 하나에 이런 표기가 여러 개 있습니다.



같은 계정, 두 가지 표기

노드에게 USDT 주소를 물어봅시다.

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

```
0:b113a994b5024a16719f69139328eb759596c38a25f59028b146fecdc3621dfe
EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs
UQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_p0p
```

- `EQ…`는 반송형(bounceable) 주소로, 보통의 컨트랙트를 이렇게 적어요.
- `UQ…`는 반송 불가형으로, 지갑은 이렇게 적기를 권해요.
- 테스트넷에서는 `kQ`와 `0Q`로 시작해요.

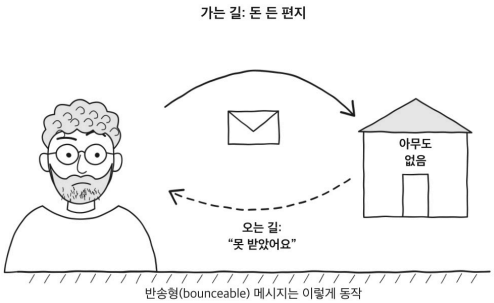
그래서 가게 데이터베이스에서는 원시 형태를 비교해요. `EQ`, `UQ` 문자열을 비교하는 것이 아닙니다.

체크섬은 오타를 잡아냅니다. 끝의 `sDs`를 `sDt`로 쓰면 노드는 422 코드로 대답해요.

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

빈집에 보낸 편지

받는 쪽이 편지를 처리하지 못했거나, 아직 컨트랙트가 없으면 어떻게 될까요? 메시지에 붙은 표시에 따라 달라요.



반송형(bounceable) 메시지는 이렇게 동작

반송형 메시지는 돌아갈 비용이 남아 있으면, 남은 돈을 담아 보낸 사람에게 돌아와요. 반송 불가형은 받는 쪽에 남습니다. 새 지갑은 아직 '배포'되지 않았을 수 있어요. 즉 지갑 프로그램이 네트워크에 적혀 있지 않은 거죠. 그래서 지갑에는 반송 불가형 주소로 돈을 보냅니다.

직접 해보기: 속도 확인하기

최신 마스터 블록 번호를 알아낸 뒤, 그 블록과 1000개 앞 블록이 나온 시각을 확인해 봅니다.

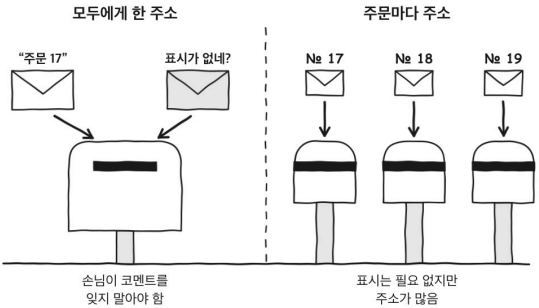
```
B=https://public-ton-mainnet.rpcnode.dev
last=$(curl -s $B/api/v2/getMasterchainInfo |
  python3 -c "import json,sys; print(json.load(sys.stdin)['result']['last']['seqno'])")
for s in $last $((last-1000)); do
  sleep 2
  curl -s "$B/api/v2/getBlockHeader?workchain=-1&shard=-9223372036854775808&seqno=$s"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

```
108011078 1795732480
108010078 1795732076
```

두 번째 숫자는 초 단위 시각이에요. 1000블록에 404초 차이이니, 블록 하나에 약 0.4초입니다.

별도 주소 대신 코멘트

TON 송금에는 글로 된 코멘트를 붙일 수 있어요. 그래서 누구의 결제인지 알아내는 방법이 두 가지입니다.



공용 주소는 손이 덜 가지만, 손님이 코멘트를 잊거나 잘못 옮겨 적을 수 있어요. 게다가 모든 지갑이 코멘트를 붙일 수 있는 것도 아닙니다. 이런 결제는 손으로 가려내야 해요.

함정: "알림이 왔으니 발송"

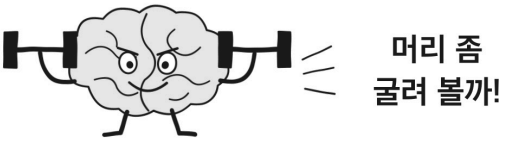
누구든 여러분의 지갑에 똑같은 알림을 보내는 컨트랙트를 만들 수 있어요. 'USDT'라는 이름의 자기 jetton을 발행할 수도 있고요. 그래서 이것들을 확인합니다.

- 알림이 내 USDT 지갑에서 왔는지. 그 주소는 진짜 대표 컨트랙트에 미리 물어봐 둡니다 (`get_wallet_address`).
- 금액이 맞는지(USDT는 소수점 아래 여섯 자리).

- 코멘트나 주소가 이 주문의 것이고, 이 결제를 아직 인정한 적이 없는지.
- 기록이 마스터체인(masterchain)에 들어갔는지, 즉 최종 확정됐는지.

알림은 보내는 사람이 알림용으로 Gram을 조금이라도 붙였을 때만 와요. 보통은 그렇게 하지만, 직접 만든 프로그램은 빠뜨릴 수 있습니다. 그러면 토큰은 알림 없이 도착해요.

생각해 보기



1. 데이터베이스에는 `UQ` 주소가 적혀 있는데, 탐색기에는 가운데가 같은 `EQ`가 보여요. 같은 계정인지 어떻게 확인할까요?
2. 사슬의 첫 트랜잭션이 성공했어요. 그럼 USDT가 도착한 걸까요?
3. 손님이 공용 주소로 결제했는데 코멘트를 잊었어요. 가게는 어떻게 해야 할까요?

정답: 1) 둘 다 원시 형태로 바꿔(`detectAddress`) 비교해요. 2) 아직 아니에요. 받는 사람의 USDT 지갑이 입금해야 해요. 3) 자동으로 인정하지 말고 금액, 보낸 사람, 송금 번호를 손으로 확인해요.

핵심 정리

- TON에서는 컨트랙트끼리 편지를 주고받아요. 지시 하나가 트랜잭션 사슬을 낳습니다.
- 계정 하나에 주소 표기가 여러 개예요. 원시 형태를 비교하세요.
- Jetton은 대표 컨트랙트와 내 jetton 지갑으로 확인해요.

다음: 20장. 다른 접근법들: Sui와 Aptos, XRP Ledger와 Stellar, Cardano, Polkadot, Cosmos.

제20장. 다른 접근법: Sui와 Aptos, XRP Ledger와 Stellar, Cardano, Polkadot, Cosmos

“XLM도 받으세요?”

한 손님은 Stellar로 결제해도 되느냐고 묻고, 다른 손님은 Sui에서 USDC를 보내고 싶어 해요. 아르티옴은 Bitcoin과 Ethereum은 이미 익혔어요. 그 지식이면 충분할까요? 꼭 그렇진 않아요. 여기는 여기만의 법이 있습니다.

알아보기: 새 네트워크에 던지는 다섯 가지 질문

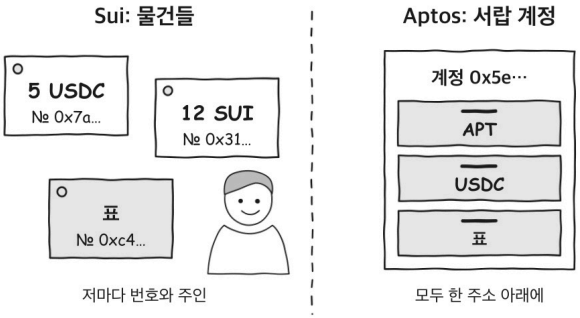
낯선 네트워크를 통째로 공부할 필요는 없어요. 처음에는 대답 다섯 개면 됩니다.

1. 돈은 어디에 있나? 계정에 있나, 따로따로인 조각에 있나?
2. 송금을 받으려면 미리 해 둬야 할 일이 있나?
3. 어느 주문의 돈인지 어떻게 아나?
4. 언제 최종 확정되나?
5. 나중에 돈을 다시 보낼 때 수수료는 무엇으로 내나?

Sui와 Aptos: 언어는 같고, 구조는 다르다

두 네트워크의 프로그램은 모두 Move 언어로 써요. 이 언어는 귀한 것이 그냥 복사되지 않도록 지킵니다. 하지만 돈을 보관하는 방식은 달라요.

- Sui는 모든 것을 객체로 저장해요. 저마다 번호가 있는 물건입니다. 돈은 따로따로인 코인 객체로 있을 수 있어요. Bitcoin의 조각과 거의 비슷하죠. 2026년부터는 주소에 합산 잔액도 있을 수 있습니다. 전체 잔액은 둘을 더한 거예요.
- Aptos는 익숙한 계정에 더 가까워요. 주소 아래에 재산이 쌓이는데, 이를 리소스라고 합니다. 계정을 미리 만들 필요는 없어요.



두 네트워크 모두 송금이 보통 1초도 안 돼 최종 확정돼요. 수수료는 자기 코인으로 냅니다. SUI 또는 APT죠. Sui에는 예외가 있어요. 2026년부터 USDC를 비롯한 일부 스테이블코인은 수수료 없이 그냥 보낼 수 있습니다(0.01 이상부터).

주소는 두 곳 모두 `0x` 와 보통 64자예요. 시작은 Ethereum과 같지만 길이가 달라요. Ethereum은 `0x` 뒤에 40자입니다.

직접 해보기: 두 네트워크에서 USDC 찾기

이 장의 네트워크에는 RpcNode의 무료 공개 노드가 없어요. 그래서 각 네트워크를 개발하는 쪽의 공개 주소에 물어볼게요. 모두가 함께 쓰고 제한이 있어서, 맛보기용이지 가게용은 아닙니다.

Sui는 GraphQL이라는 방식이에요(필요한 필드를 나열합니다).

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

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

Aptos는 11장의 평범한 REST예요.

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

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

여기서도 저기서도 토큰은 'USDC'라는 단어가 아니라, 발행사인 Circle의 목록에 있는 긴 주소로 찾아요. 9장의 원칙입니다. 소수점 아래는 두 경우 모두 여섯 자리예요.

옛 예제는 조심하세요. 예전에는 Sui에 JSON-RPC로 물었어요. 2026년 7월에 Sui 재단의 노드에서는 이를 껐고, 지금은 gRPC와 GraphQL이 있습니다. 글에서 예제를 복사하기 전에 날짜를 보세요.

XRP Ledger와 Stellar: 송금의 네트워크

두 네트워크 모두 오래전부터 자기 코인을 보내는 것뿐 아니라 남의 토큰도 다룰 수 있어요. 체인의 한 페이지를 여기서는 ledger(원장)라고 합니다. XRP Ledger는 3–5초마다, Stellar는 약 5초마다 새 원장을 확정해요. 확정된 것은 다시 쓰지 않습니다.

그런데 낯선 규칙이 두 가지 있어요.

첫째, 자리 보증금. 최소 금액, 즉 reserve(예치금)가 들어 있지 않으면 계정은 존재하지 않아요. 그리고 USDC 같은 토큰을 받으려면 계정 주인이 미리 신뢰선(trust line)을 열어야 합니다. "네,

이것을 보유하는 데 동의합니다"라는 뜻이죠. 이런 항목이 하나 늘 때마다 예치금이 늘어요.

	XRP Ledger	Stellar
주소의 시작	`r`	`G`
계정 최소 금액	1 XRP	1 XLM
trust line당 추가	0.2 XRP(처음 두 개는 없어도 될 수 있음)	0.5 XLM
최소 수수료	0.00001 XRP	작업 하나당 0.00001 XLM
주문 번호	destination tag	memo 또는 `M` 주소

이 숫자들은 네트워크가 투표로 바꿀 수 있어서 노드에게 물어봅니다.

```
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`는 0.00001, `1E0`은 1, `2E-1`은 0.2예요. 이번에는 Stellar입니다.

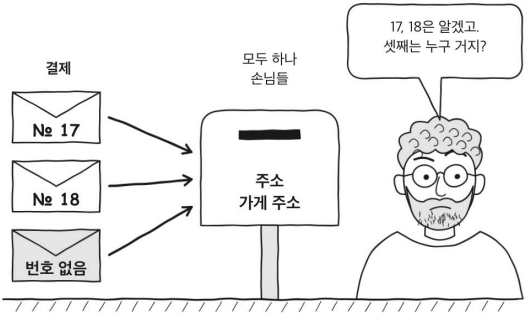
```
curl -s 'https://horizon.stellar.org/ledgers?order=desc&limit=1' \
  | grep -E '"(base_fee|base_reserve)_in_stroops"'
```

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

1 XLM은 1,000만 stroops예요. 그러니 수수료는 0.00001 XLM이고, 예치금 한 몫은 0.5 XLM입니다(계정에는 최소 두 몫이 필요해요).

모두에게 하나의 주소

두 번째 규칙. 여기서는 주문마다 주소를 따로 만들면 부담이 커요. 주소마다 예치금이 묶이니까요. 그래서 가게는 주소 하나를 보여 주고 번호를 적어 달라고 합니다. XRP Ledger의 destination tag, Stellar의 memo입니다.

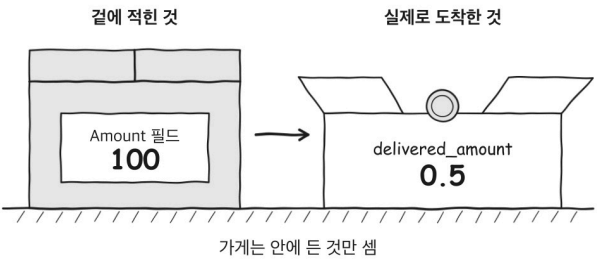


손님이 번호를 잊으면 돈은 왔는데 무엇의 값인지 알 수 없어요. 도움이 되는 방법은 이렇습니다.

- XRP Ledger에서는 `RequireDest` 설정을 켤 수 있어요. 번호 없는 송금은 네트워크가 통과시키지 않습니다.
- 번호를 주소 안에 숨길 수도 있어요. XRP Ledger의 X-주소, Stellar의 `M` 주소입니다. 손님의 지갑이 알아들을 때의 이야기예요.

함정: 라벨에는 100, 안에는 0.5

XRP Ledger에서는 보내는 사람이 부분 결제(partial payment)를 허용할 수 있어요. 덜 도착해도 송금은 성공으로 칩니다. 이때 `Amount` 필드는 "도착한 금액"이 아니라 "최대 금액"을 뜻해요. 실제로 얼마가 도착했는지는 `delivered_amount`에 적혀 있습니다.



네트워크에 있는 진짜 기록이에요. 해시로 물어봅시다.

```
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"}
```

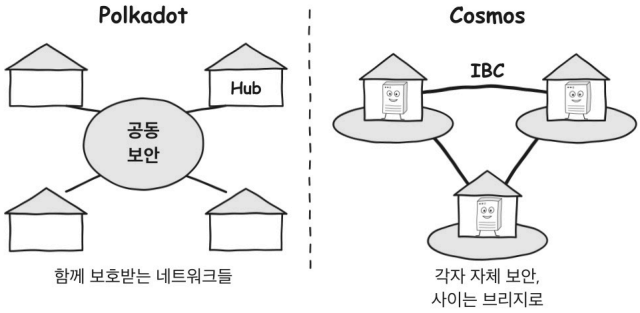
결과는 `tesSUCCESS`이고, `Amount`는 약 9.55인데 도착한 것은 약 4.53이에요. 이런 기록 자체가 사기는 아닙니다. 하지만 사기꾼은 큰 금액을 적어 놓고 부스러기만 보낸 뒤 상품을 기다릴 수 있어요. `delivered_amount`만 세고, 어떤 토큰이고 어느 발행사의 것인지 확인하세요.

Cardano: 짐이 달린 조각

Cardano의 구조는 Bitcoin(14장)에 더 가까워요. 잔액이 아니라 쓰지 않은 조각들이 있죠. 다만 조각이 할 수 있는 일이 더 많아서, 이 모델을 확장 UTXO(eUTXO)라고 합니다.

- 조각 안에는 ADA 코인과 함께 다른 토큰들이 들어 있어요. 토큰을 위한 별도의 컨트랙트는 필요 없어요.
- 토큰은 혼자 여행하지 않아요. 같은 조각 안에 ADA가 조금은 꼭 있어야 합니다. 그래서 토큰을 받으면 코인도 약간 함께 와요.
- 수수료는 송금의 바이트 크기에 달려 있어요.
- 확신은 서서히 커져요. 위에 블록이 많을수록 믿을 만합니다.

Polkadot과 Cosmos: 하나가 아니라 여러 네트워크



Polkadot은 자기에게 연결된 네트워크들을 지켜 주는 중심 체인이에요. 중요한 변화가 있었어요. 2025년 11월부터 사용자의 잔액은 중심 체인이 아니라 Polkadot Hub(Asset Hub라고도 해요) 네트워크에 있습니다.

- USDT(자산 번호 1984)와 USDC(번호 1337)도 그곳에 있어요.
- 계정에는 생존 기준선, existential deposit이 있어요. 0.01 DOT입니다. DOT도, 필요한 만큼의 USDT나 USDC도 모자라면 계정은 지워져요.
- 주소는 Bitcoin의 옛 주소처럼 `1`로 시작해요. 헷갈리지 마세요.

Cosmos는 공통 부품으로 만든 독립 네트워크들이에요. 저마다 자체 보안, 코인, 주소 접두어가 있습니다(Cosmos Hub는 `cosmos1…`). 블록은 빨리 최종 확정돼요. 송금에는 주문 번호를 적을 memo 필드가 있어요.

네트워크들은 IBC 프로토콜로 이어져요. 이웃 네트워크에서 건너온 토큰은 익숙한 단어가 아니라 `ibc/`와 긴 도장으로 불립니다. 예를 들어 `ibc/7F1D3FCF…`. 도장은 어떤 길로 왔는지를 말해 줘요.

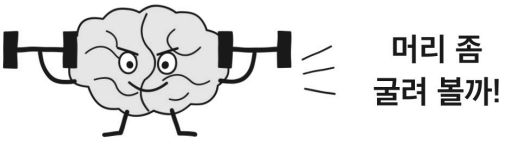
그리고 스테이블코인이 떠날 수도 있어요. Circle은 Cosmos 계열의 Noble 네트워크에서 USDC 발행을 중단한다고 발표했어요. 2026년 10월 13일부터 신규 발행이 꺼졌고, 2027년 1월 12일에는 그곳의 USDC가 정지됩니다. 보유자들에게는 미리 옮기라고 권하고 있어요.

한눈에 보기

네트워크	주소	미리 필요한 것	최종 확정
Sui	`0x…`	없음	1초 미만
Aptos	`0x…`	없음	1초 미만
XRP Ledger	`r…`	예치금, trust line	3–5초
Stellar	`G…`	예치금, trust line	약 5초
Cardano	`addr1…`	없음	서서히
Polkadot Hub	`1…`	기준선 이상의 잔액	표시로 확인
Cosmos Hub	`cosmos1…`	없음	빠름

Trust line은 토큰에만 필요해요. 2026년 11월 27일에 확인했으며, 규칙은 바뀝니다.

생각해 보기



1. 아르티옴이 Stellar에 텅 빈 새 주소를 만들고 곧바로 USDC 청구서를 냈어요. 무엇이 잘못이고, XLM은 최소 얼마가 필요할까요?
2. 송금이 왔어요. `Amount`는 100 USDC, `delivered_amount`는 0.5, 결과는 `tesSUCCESS`. 100어치 커피를 보낼까요?
3. XRP Ledger 손님들이 destination tag를 자꾸 잊어요. 무엇을 켜야 할까요?

정답: 1) 계정이 아직 없고 trust line도 열지 않았어요. 예치금으로 최소 1.5 XLM과 약간의 수수료가 필요해요. 2) 아니요, 도착한 것은 0.5예요. 3) RequireDest 설정.

핵심 정리

- 낯선 네트워크에는 손님에게 주소를 보여 주기 전에 다섯 가지 질문을 하세요.
- XRP Ledger와 Stellar에는 예치금, trust line, 주문 번호가 필요해요. XRP Ledger에서는 `delivered_amount`만 세세요.
- Ethereum에서 든 습관과 옛 예제는 다른 네트워크로 가져갈 수 없어요.

다음: 21장. 어떤 네트워크를 고를까.

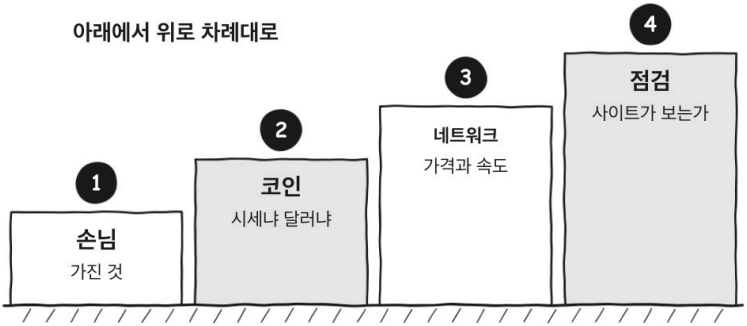
제21장. 어떤 네트워크를 고를까

"그래서 이 중에 뭘 골라야 하죠?"

일곱 장을 지나는 동안 이름이 잔뜩 쌓였어요. Bitcoin, Ethereum, TRON, Solana, TON, 그리고 열 개쯤 더. 아르티옴에게는 온라인 가게 하나와 커피 로스팅 작업장 하나가 있어요. 전부 연결할까? 하나만 고를까? 무엇을 기준으로?

알아보기: 네 단계

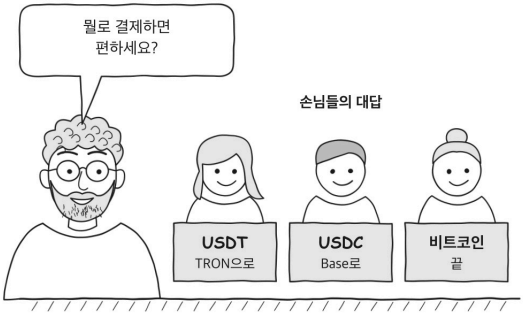
'가장 좋은 네트워크'는 없어요. 여러분의 손님과 주문에 맞는 네트워크가 있을 뿐이에요. 순서대로 가면 편합니다.



여기서 다루는 것은 무엇으로 결제를 받으면 편한가예요. 어디에 돈을 투자하라는 조언이 아닙니다.

1단계. 손님에게 물어보기

결제하는 사람은 이미 코인을 가진 사람이에요. 손님의 코인은 한 네트워크에 있는데 여러분이 다른 네트워크로 받으면, 결제는 아마 이루어지지 않을 거예요.



소식지에서, 채팅에서, 주문할 때 물어보세요. 질문 두 개면 충분해요. “무엇을 가지고 계세요?” 그리고 “어느 네트워크에 있나요?”

2단계. 어떤 코인

	네트워크의 기본 코인	스테이블코인 (USDT, USDC)
달러 가격	변함, 때로는 급격히	1달러 언저리를 유지하려 함
누가 주인인가	네트워크의 규칙	발행사가 있고, 동결할 수 있음(9장)
수수료는 무엇으로	그 코인 자체	보통 스테이블코인이 아니라 네트워크의 코인

아르티옴의 커피 값이 달러이거나 달러에 묶여 있다면, 스테이블코인이 뜻밖의 일을 덜 만들어요. 하지만 이것도 믿음이에요. 발행하는 회사에 대한 믿음입니다.

3단계. 어느 네트워크

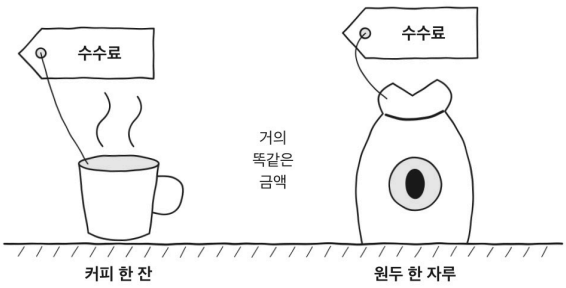
같은 스테이블코인이 어디에나 사는 것은 아니에요. 14–20장의 네트워크를 모았습니다. '예'는 발행사 사이트에 이 네트워크용으로 바로 그 코인의 주소가 적혀 있다는 뜻이에요.

네트워크	수수료	Tether의 USDT	Circle의 USDC	잊지 말 것
Bitcoin	BTC	아니요	아니요	컨펌
Litecoin, Doge 등	자체 코인	아니요	아니요	서로 다른 네트워크
Ethereum	ETH	예	예	가스
Base, Arbitrum	ETH	아니요	예	finalized
BNB Smart Chain	BNB	아니요	아니요	발행사
TRON	TRX	예	아니요	에너지
Solana	SOL	예	예	토큰 계정
TON	Gram	예	아니요	메시지
Aptos	APT	예	예	긴 주소
Sui	SUI	아니요	예	gRPC, GraphQL
Polkadot Hub	DOT	예	예	계정 기준선
XRP Ledger	XRP	아니요	예	예치금, 태그
Stellar	XLM	아니요	예	예치금, 태그
Cardano	ADA	아니요	아니요	토큰에 딸린 ADA

2026년 11월 27일 기준 tether.to/en/supported-protocols와 developers.circle.com에 따른 것이에요. '아니요'가 그 이름의 토큰이 거기에 없다는 뜻은 아닙니다. 누가 발행하는지 확인하세요(9장). Cosmos는 개별 네트워크에 따라 다릅니다.

송금 비용과 주문 크기

수수료는 금액이 아니라 송금의 '무게'와 네트워크의 혼잡도에 달려 있어요(7장).



원두 한 자루에는 푼돈이에요. 커피 한 잔에는 수수료가 잔 값보다 비쌀 수 있어요. 그래서 작은 주문은 어디가 송금이 싼지 봅니다. 가격은 바뀌니, 옛날 표가 아니라 보내기 직전에 지갑에서 확인해요.

그리고 돌아오는 길도 기억하세요. 받은 것을 다시 보내려면 그 주소에 수수료용 코인이 있어야 해요. 주소가 백 개면 수수료 내는 송금도 백 번입니다.

얼마나 기다릴까

최종 확정까지	네트워크
몇 초 이내	BNB Smart Chain, Solana, TON, Aptos, Sui, XRP Ledger, Stellar
약 1분	TRON
수십 분	Ethereum, Base, Arbitrum One
약 1시간	Bitcoin (6컨펌)

8장과 17–20장을 바탕으로 한 대략의 값이에요.

직접 해보기: 확정 블록은 얼마나 오래됐나

EVM 네트워크들에게 `finalized` 블록(8장)을 물어보고, 그 블록이 몇 초 전에 나왔는지 계산해 봅시다.

```
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 ))초"
  sleep 1.5
done
```

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

Ethereum은 약 15분, Base는 약 22분이에요. BNB Smart Chain은 1초입니다. 여러분의 숫자는 다를 거예요. 표시가 뚝뚝 끊기며 움직이니까요. 작은 주문 때문에 가장 확실한 표시까지 꼭 기다릴 필요는 없지만, 그때의 위험은 여러분이 집니다.

4단계. 내 사이트가 알아볼 수 있을까

사이트에 주소를 내걸기 전에 확인하세요.

- 제공업체에 이 네트워크용 엔드포인트가 있는지(13장).

- 바로 이 네트워크에 대한 결제 알림이 있는지. 둘은 같은 것이 아니에요. RpcNode는 이벤트를 위해 블록을 분석하는 네트워크보다 더 많은 네트워크의 엔드포인트를 내줍니다(26장과 27장).
- 진짜 돈 없이 시험해 볼 테스트넷이 있는지.
- 내 지갑과 코인을 환전하는 곳이 바로 이 네트워크의 이 코인을 다룰 수 있는지.

그리고 요금제도 들여다보세요. RpcNode의 Free에서는 Ethereum과 Solana의 엔드포인트만 만들 수 있어요(22장).

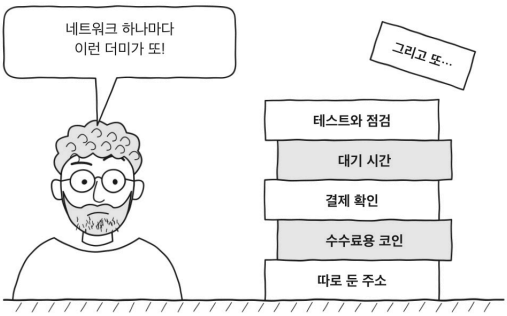
예: 아르티옴의 결정

손님 20명에게 물었다고 해 볼게요. 숫자는 연습용이고, 여러분 가게는 다를 거예요.

무엇으로 결제하고 싶은지	인원	결정
TRON의 USDT	9	바로 연결
Base의 USDC	5	바로 연결
BTC	3	요청이 있으면 나중에
ETH	2	나중에
XLM	1	아직은 아님

네트워크 두 개로 20명 중 14명이 해결돼요. 나머지에게 아르티옴은 아직 모든 것을 받지는 않는다고 솔직히 알립니다.

함정: “한꺼번에 다 연결하지 뭐”

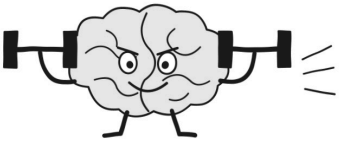


결제 페이지에 줄 하나 더 넣는 건 공짜처럼 보여요. 하지만 그 뒤에는 따로 있는 주소, 따로 필요한 수수료 코인, 따로 정해야 할 확인 규칙, 업데이트 뒤의 뜻밖의 일들이 있습니다. 한두 개로 시작하고, 요청이 있으면 늘리세요.

두 번째 함정: 결제 페이지에 그냥 ‘USDT’라고만 적는 것. 손님이 네트워크를 알아서 고르다가 틀린 것을 고를 수 있어요(16장). 코인과 네트워크를 함께, 크게 적으세요. “TRON 네트워크의

USDT".

생각해 보기



1. 주문은 5달러인데, 고른 네트워크의 오늘 수수료가 2달러예요. 누가 불편하고, 무엇을 해야 할까요?
2. 손님이 Base의 USDT를 받아 달라고 해요. Tether 사이트의 목록은 뭐라고 할까요?
3. 아르티옴이 첫날부터 네트워크 여덟 개를 받으려 해요. 뭐라고 말려야 할까요?

정답: 1) 손님이에요. 수수료가 주문 금액의 40%만큼 더 붙어요. 지금 송금이 더 싼 네트워크를 제안하세요. 2) 거기에 Base는 없어요. 그 이름의 토큰을 받기 전에 누가 발행하는지 알아보세요. 3) 걱정거리 더미가 여덟 개예요. 손님에게 가장 필요한 한두 개로 시작하세요.

핵심 정리

- 선택은 순서대로 해요. 손님, 코인, 네트워크, 점검.
- 수수료를 주문 크기와 비교하고, 기다리는 시간도 따지세요.
- 한두 네트워크로 시작하고, 코인은 늘 네트워크와 함께 말하세요.

파트 4 끝. 아르티옴은 네트워크들이 어떻게 다른지 알았고, 무엇부터 시작할지 정했어요. 다음: 파트 5, 22장. RpcNode에서의 첫 5분.

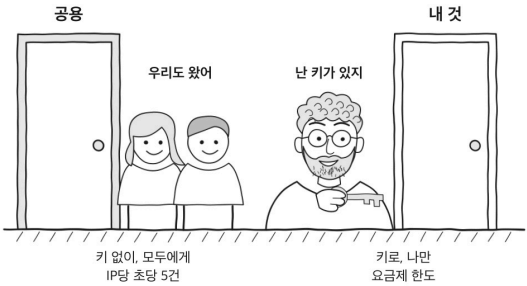
제22장. RpcNode에서의 첫 5분

"내 문을 갖고 싶어요"

아르티옴은 책 내내 공개 노드의 문을 두드렸어요. 가입 없이도 대답해 줬지만, 그 문은 모두의 것이에요. 가게에는 자기 문이 필요합니다. 만드는 데 오래 걸릴까? 뭘 채워 넣어야 할까?

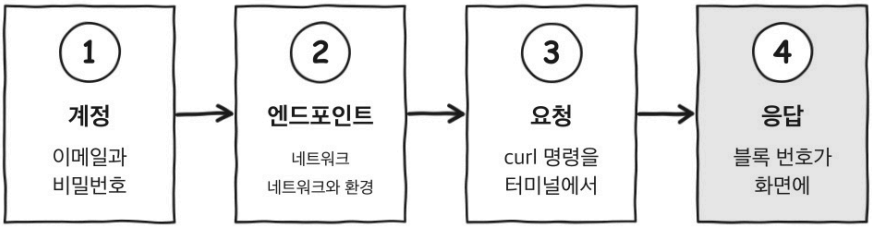
알아보기: 공용 문과 내 문

프로그램이 노드에게 질문하는 주소를 엔드포인트라고 해요(3장). RpcNode에는 두 종류가 있습니다.



	공개 주소	개인 엔드포인트
가입	필요 없음	필요함
키	없음	주소에 들어 있음
한도	IP당 초당 5건	요금제에 따라
무거운 메서드	막힘	열림
아카이브(24장)	막힘	유료 요금제에서
WebSocket과 gRPC	없음	네트워크가 지원하는 곳에서

내 엔드포인트는 네 단계면 생겨요.



1단계. 가입

`cabinet.rpcnode.dev/register`를 열거나 rpcnode.dev 사이트에서 '시작하기'를 누르세요. 대시보드는 서비스 안의 내 페이지예요. 엔드포인트, 사용량, 요금제가 여기에 있습니다.

1. 이메일을 적고, 비밀번호를 정해 한 번 더 입력하세요. 이름은 안 적어도 돼요.
2. '다음에 동의합니다: 이용약관'을 읽고 체크하세요. "로봇이 아닙니다" 확인이 나오면 통과하세요.
3. '계속'을 누르세요.
4. 메일로 일회용 코드가 옵니다. 입력하고 '확인 후 계속'을 누르세요.

지름길도 있어요. Google이나 GitHub로 로그인하는 거예요.

새 계정에는 무료 요금제 Free가 적용되고, 카드는 필요 없어요. 월 크레딧 1,500만, 초당 요청 15건, 엔드포인트 3개, 네트워크는 Ethereum과 Solana 두 개입니다. 크레딧이 무엇인지는 25장에서 살펴봐요.

2단계. 엔드포인트

대시보드 메뉴에서 '엔드포인트'를 고르고 '새 엔드포인트'를 누르세요. 세 단계로 된 창이 열려요.

창의 단계	할 일
네트워크 선택	예를 들어 Ethereum
환경 선택	메인넷 또는 테스트넷(아래에서 설명)
확인	살펴보고 '만들기' 누르기

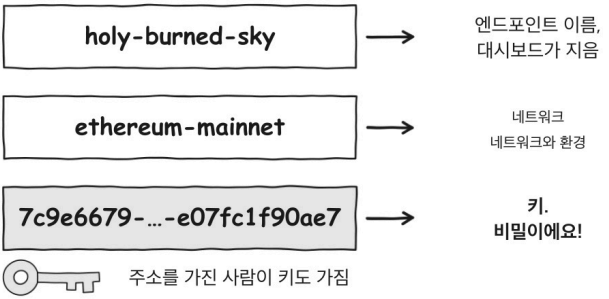
환경은 네트워크의 버전이에요. 진짜인지 연습용인지요. 엔드포인트 이름은 대시보드가 영어 단어 세 개로 알아서 지어 줍니다.

Free 요금제에서는 만들기 전에 아카이브 노드에 관한 안내가 보여요. 오류가 아니고, 24장에서 설명합니다.

주소는 무엇으로 이루어졌나

RpcNode 문서에 나오는 예예요. 원래 한 줄인데, 여기서는 빗금 앞에서 줄을 바꿨습니다.

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



빗금 뒤의 긴 꼬리는 UUID라고 해요. 사실상 알아맞힐 수 없는 번호입니다. 이것이 바로 키예요. 요청에 별도의 비밀번호를 붙일 필요는 없습니다.

주소는 손으로 치지 말고 대시보드의 버튼으로 복사하세요. 화면에는 키의 일부가 가려져 있지만, 복사하면 전체가 복사돼요.

메뉴에는 'API 키'도 있어요. 이건 다른 것입니다. "결제를 만들어"처럼 서비스 자체에 내리는 명령에 서명할 때 써요. 노드에게 하는 보통의 질문에는 필요 없습니다.

3단계. 직접 해보기: 첫 요청

주소를 설정에 넣고 마지막 블록 번호를 물어볼게요. 이것은 대강의 모습이니, 글자 자리에 여러분의 주소를 넣으세요.

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

개인 주소는 여러분에게만 있으니, 책에서는 같은 명령을 공개 주소로 실행한 응답을 보여드려요. `RPC_URL='https://public-ethereum-mainnet.rpcnode.dev'` 입니다.

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

`0x19464e4` 는 26,502,372번 블록이에요. 3장에서 노드는 26,502,035라고 했죠. 그사이 네트워크는 337블록을 더 쌓았어요.

Solana라면 `eth_blockNumber` 대신 `getSlot` 이라고 쓰세요.

대시보드의 '개요' 페이지에는 '시작하기' 목록이 있어요. 거기에 여러분의 주소가 이미 들어간 완성된 명령과 "첫 요청을 기다리는 중…"이라는 글이 있습니다. 그리고 엔드포인트 페이지의 '실시간' 탭에서는 요청 하나하나가 보여요. 시각, 메서드, 응답, 지연 시간입니다.

잘 안 됐다면

일부러 0으로만 된 가짜 키로 물어볼게요.

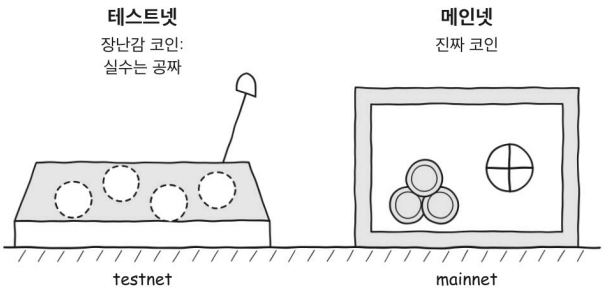
```
HOST=https://holy-burned-sky-ethereum-mainnet.rpcnode.dev
KEY=00000000-0000-0000-0000-000000000000
curl -s -w '  HTTP %{http_code}\n' $HOST/$KEY \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
```

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

`endpoint not found`는 "그런 엔드포인트는 없다"는 뜻이에요. 주소가 통째로 복사됐는지 확인하세요. 틀린 주소를 반복문으로 계속 보내지 마세요. 끈질기게 두드리면 한동안 아예 들어주지 않습니다.

테스트넷: 장난감 돈의 연습장

테스트넷(testnet)은 진짜 네트워크와 같은 규칙으로 돌아가지만, 코인에는 아무 값어치가 없어요. 진짜 네트워크는 mainnet(메인넷)이라고 합니다. 여기서는 공짜로 실수할 수 있어요.



테스트 코인은 '수도꼭지'(faucet)에서 받아요. 적어 준 주소로 코인을 보내 주는 페이지입니다.

네트워크	RpcNode의 테스트 환경 예
Ethereum	Sepolia, Hoodi
Solana	testnet, devnet
Bitcoin	testnet4, signet
TRON	Nile, Shasta

Ethereum 애플리케이션이라면 ethereum.org 사이트는 Sepolia를 권해요. Hoodi는 네트워크 자체와 검증자를 시험하는 사람에게 필요합니다.

가입 없이 쓰는 테스트넷도 있어요. 2026년 11월 27일 기준 공개 노드 목록에는 Bitcoin testnet4, TRON Nile, TON testnet이 있습니다. 진짜 TRON과 테스트 TRON에게 네트워크 번호(16장)를 물어볼게요.

```
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"}
```

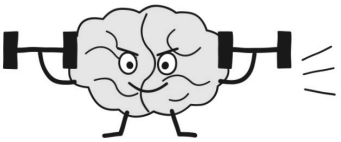
번호가 달라요. 서로 다른 두 장부입니다. 엔드포인트도 네트워크 하나와 환경 하나에 묶여 있어요. 테스트넷에서 진짜로 넘어가려면 새로 만듭니다.

함정: “주소를 웹페이지에 바로 넣자”

페이지 코드에 들어간 것은 어떤 방문자든 볼 수 있어요. 그 사람이 여러분의 주소를 가져다가 여러분의 한도를 쓸 겁니다.

- 손님의 브라우저가 아니라 내 서버에서 노드에게 물으세요.
- 주소는 코드나 공유 코드 저장소가 아니라 서버 설정에 두세요.
- 엔드포인트 페이지의 ‘설정’에서 ‘IP 허용 목록’을 지정하세요. 그러면 다른 컴퓨터는 거절당해요.
- ‘허용 Origin’은 남의 사이트에서 브라우저로 엔드포인트를 호출하지 못하게 해요. 하지만 브라우저 밖의 프로그램은 이 목록이 막지 못합니다.
- 주소가 새어 나갔나요? 새것을 만들어 사이트를 갈아 끼우고, 옛것은 삭제하세요. 되살릴 수는 없어요.

생각해 보기



머리 좀 굴려 볼까!

1. 아르티옴이 블록 번호를 보여 주려고 가게 페이지의 스크립트에 엔드포인트 주소를 적었어요. 누가 키를 보게 될까요?
2. Sepolia에서 모든 확인을 마쳤어요. 같은 엔드포인트로 진짜 결제를 받아도 될까요?
3. `endpoint not found`가 와요. 어디서부터 찾아볼까요?

정답: 1) 모든 방문자. 2) 안 돼요. mainnet 엔드포인트가 필요해요. 3) 주소를 대시보드의 것과 대조하고, 처음부터 통째로 다시 복사해요.

핵심 정리

- 개인 엔드포인트는 안에 키가 든 주소예요. 비밀번호처럼 간수하세요.
- 엔드포인트는 네트워크 하나와 환경 하나에 묶여 있어요.
- 먼저 테스트넷, 그다음 진짜.

다음: 23장. 요청에는 무슨 일이 벌어질까. 게이트웨이, 노드 선택, 그리고 세 지역.

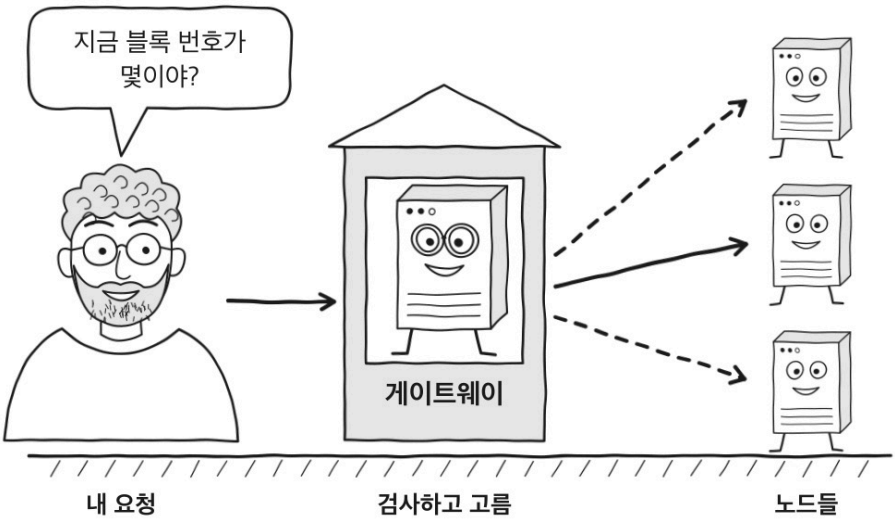
제23장. 요청에는 무슨 일이 벌어질까

“그런데 누가 대답한 거죠?”

아르티옴은 요청을 보내고 블록 번호를 받았어요. 그런데 질문은 어디로 갔을까요? 어느 노드가 대답했을까요? 그리고 한창 세일 중에 그 노드가 고장 나면 어떻게 될까요?

알아보기: 입구에는 게이트웨이가 있다

여러분의 요청은 노드로 바로 가지 않아요. 먼저 게이트웨이에 도착합니다. 요청을 받고, 검사하고, 알맞은 노드에 넘겨주는 배차 담당 프로그램이에요.



게이트웨이가 길목에서 확인하는 것

회사 정문의 출입 검사와 비슷해요. 검사 하나를 통과하지 못하면 요청은 더 가지 못합니다.

검사	통과하지 못하면
그런 엔드포인트가 있는지, 주소가 맞는지	HTTP 404, `-32000`
꺼져 있지 않은지, 내 IP와 사이트가 허용됐는지	HTTP 403, `-32000`
너무 자주 묻지 않는지	HTTP 429, `-32005`
이 네트워크에 그런 메서드가 있는지	HTTP 404, `-32601`
크레딧이 충분한지	HTTP 402, `-32006`

네트워크마다 자기 메서드 목록이 있어요. 목록에 없는 것을 물어볼게요.

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

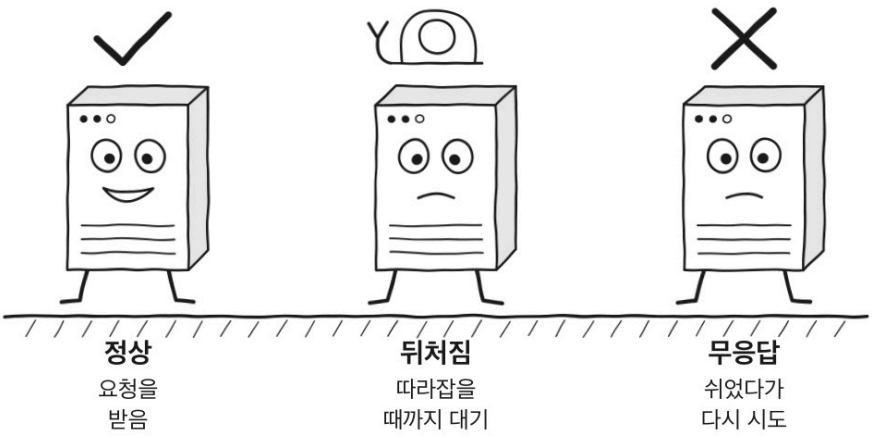
```
{"jsonrpc":"2.0","id":1,"error":{"code":-32601,
"message":"method not found: eth_noSuchMethod"}}  HTTP 404
```

이 질문은 노드까지 가지 않았어요. 게이트웨이가 직접 대답했습니다.

노드 고르기: 부하 분산

네트워크 하나에 노드가 여러 대 있을 수 있어요. 이들 사이에 요청을 나눠 주는 것을 부하 분산(로드 밸런싱)이라고 합니다. 가게의 계산대와 비슷해요. 새 손님은 한가한 쪽으로 보내죠.

먼저 게이트웨이는 노드가 멀쩡한지 봅니다.



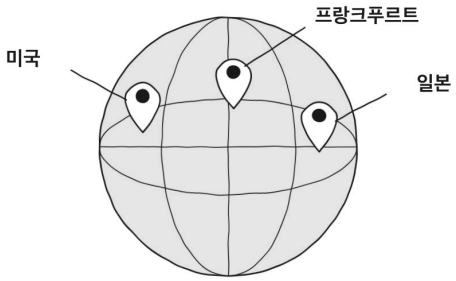
- 뒤처진 노드는 씩씩하게 대답하지만 어제 이야기를 해요(12장). 최신 노드가 있는 동안에는 요청이 그쪽으로 갑니다.

- 건강한 노드 가운데 바쁜 쪽은 덜, 한가한 쪽은 더 받아요.
- 연결이 끊기거나 노드가 내부 오류를 돌려주면, 같은 요청이 다음 노드로 가요. 여러분은 눈치채지 못합니다.
- 연달아 몇 번 해내지 못한 노드는 한동안 쉬게 두었다가, 다시 시도해요.

단서가 하나 있어요. 건강한 노드가 하나도 남지 않으면, 게이트웨이는 그래도 뒤처진 노드를 시도합니다. 아예 대답이 없는 것보다 옛 대답이 나을 때도 있으니까요. 그러니 신선도가 중요한 곳에서는 블록 시각을 직접 확인하세요(12장).

세 지역

RpcNode는 세 지역에서 돌아가요. 프랑크푸르트, 미국, 일본입니다. 노드가 가까울수록 오가는 길이 짧아요. 게이트웨이는 자기에게 가장 가까운 건강한 노드를 고르려 하고, 먼 노드는 예비로 둡니다.



RpcNode의 세 지역 (그림일 뿐, 지도 아님)

모든 네트워크에 세 곳 전부 노드가 있는 것은 아니에요. 필요한 네트워크가 어디서 어떻게 대답하는지는 `rpcnode.dev/status` 페이지에서 볼 수 있습니다.

직접 해보기: 봉투의 표시 읽기

응답과 함께 헤더가 와요. 봉투에 찍힌 도장 같은 업무용 표시입니다. 그중 두 개는 게이트웨이와 노드가 어느 나라에 있었는지 알려 줘요.

```
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`는 독일, `JP`는 일본이에요. 이번에는 모든 요청을 독일의 게이트웨이가 받았고, Base와 Arbitrum은 일본까지 다녀왔어요. 여러분에게는 다른 글자가 나올 수 있습니다. 업무용 표시이니, 이것에 기대어 사이트를 만들지 마세요.

노드의 오류도 대답이다

USDT 컨트랙트에게 물어볼게요. 텅 빈 주소에서 어마어마한 금액을 보내면 어떻게 될까요? `eth_call` 메서드는 아무것도 보내지 않고 시험만 해요. `data` 필드에는 송금 함수 번호, 받는 사람(`TO`), 금액(`SUM`)이 이어 붙어 있습니다.

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

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

이 오래된 컨트랙트가 "안 돼"라고 말하는 방식이에요. 노드는 제 일을 다 했습니다. 프로그램이 멈췄다고 솔직하게 알렸으니까요. 다른 노드도 똑같이 대답할 테니, 게이트웨이는 이런 대답을 HTTP 200과 함께 그대로 전해 줍니다.

아무도 대답하지 않을 때

더 나쁜 경우도 있어요. 게이트웨이가 노드들을 다 돌았는데 대답이 없는 거죠. 그러면 `-32002` 코드, `upstream unavailable`("노드를 사용할 수 없음")이라는 문구, 500번대 HTTP가 옵니다. 이런 호출의 크레딧은 돌려줘요.

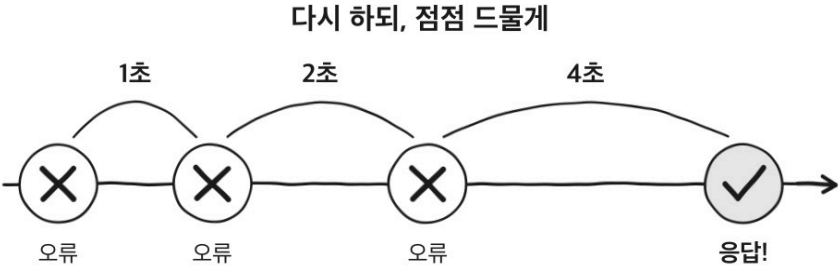
그게 돈을 보내는 요청이라면?

노드가 트랜잭션 전송을 거절했다면, 그것이 대답이에요. 하지만 노드가 아예 반응하지 않았다면, 게이트웨이는 서명된 같은 송금을 다른 노드에 넘길 수 있습니다.

그래도 돈이 두 번 나가지는 않아요. 서명된 트랜잭션은 해시가 같고, 네트워크는 한 번만 받으니까요. 다만 대답을 받지 못한 쪽이 여러분이라면, 무턱대고 새 트랜잭션에 서명하지 마세요. 먼저 옛것을 해시로 찾아보세요(7장).

직접 해보기: 똑똑하게 다시 보내기

게이트웨이는 노드를 바꿔 가며 시도하지만, 여러분과 게이트웨이 사이의 연결도 말썽을 부릴 수 있어요. 직접 다시 보내되, 쉬는 시간을 늘려 가세요. 1, 2, 4, 8초. 영어로 backoff(백오프)라고 합니다.



`ask.js` 파일이고, `node ask.js`로 실행해요(Node.js 필요).

```
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('연결 안 됨')
    }
    await sleep(pause * 1000)
  }
  throw new Error('물어보지 못했어요')
}

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

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

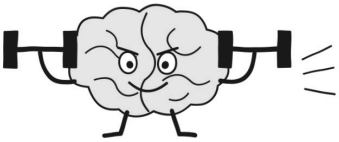
두 번째에는 될 수 있는 것만 다시 보내요. “너무 자주”, 끊김, 500번대. 요청 자체의 오류는 다시 보내 봐야 소용없습니다. 그리고 응답을 받은 뒤 `error` 필드 확인을 잊지 마세요(11장).

함정: “대답 없음”은 “결제 없음”이 아니다

물어보지 못했다면, 사이트는 손님이 결제했는지 모르는 거예요. “결제되지 않았습니다”가 아니라 “결제를 확인하는 중입니다”라고 쓰고, 나중에 다시 물으세요.

그리고 기억하세요. 노드가 여러 대인 제공업체는 노드 한 대보다 덜 죽지만, '절대' 안 죽는 것은 아닙니다(3장).

생각해 보기



1. HTTP 200과 `execution reverted`가 왔어요. 요청을 다섯 번 다시 보내면 도움이 될까요?
2. 프로그램이 `eth_blokNumber`를 물었어요. 요청이 노드까지 갔을까요?
3. 아르티옴이 송금을 보냈는데 대답을 받지 못했어요. 무엇부터 해야 할까요?

정답: 1) 아니요. 이건 장애가 아니라 진짜 대답이에요. 2) 아니요, 게이트웨이가 모르는 메서드를 거절했어요. 3) 해시로 트랜잭션을 찾아봐요.

핵심 정리

- 게이트웨이는 요청을 검사하고 건강한 노드를 골라요. 하나가 대답하지 않으면 다른 노드를 시도해요.
- 요청에 대한 노드 자체의 오류는 그대로 여러분에게 와요.
- 그래도 여러분의 프로그램은 쉬어 가며 다시 보낼 줄 알아야 해요.

다음: 24장. 기록과 아카이브 데이터.

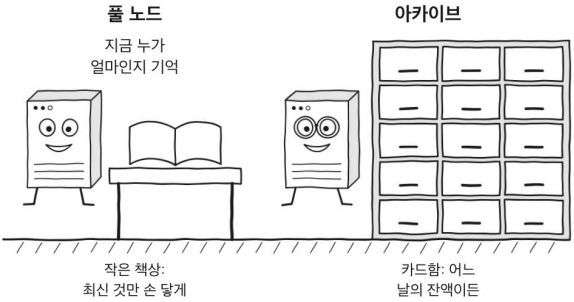
제24장. 기록과 아카이브 데이터

"1월 1일에는 얼마였죠?"

회계사가 아르티옴에게 부탁해요. "올해 초에 가게 주소에 얼마가 있었는지 알려 주세요." 오늘에 대해서는 노드가 바로 대답하는데, 그때에 대해서는요?

알아보기: 기록과 잔액

10장을 떠올려 보세요. 히스토리는 블록과 송금이에요. 상태는 특정 순간에 누가 무엇을 가졌는지입니다.



풀 노드는 최근 잔액을 손 닿는 곳에 둬요. 아카이브 노드는 어느 블록의 잔액이든 기억하는 대신 디스크가 훨씬 많이 필요합니다.

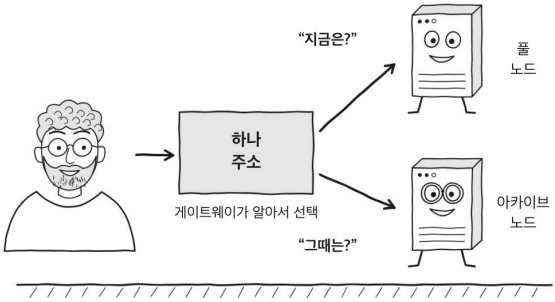
어떤 질문에 아카이브가 필요할까

질문	예
옛 블록에서의 잔액이나 컨트랙트의 대답	블록 번호를 지정한 `eth_getBalance`, `eth_call`
지난 블록의 이벤트	`eth_getLogs`
트랜잭션을 단계별로 '되감기'(트레이싱)	`trace_…`, `debug_…`
상태에 대한 증명	옛 블록의 `eth_getProof`
오래된 Solana 슬롯	옛 번호의 `getBlock`

"지금 계정에 얼마 있어?"와 보통의 최근 결제에는 아카이브가 필요 없어요. 가게의 일상 업무에는 대개 필요하지 않습니다.

RpcNode에서는 이렇게

엔드포인트 주소는 하나예요. 노드를 직접 고를 필요는 없습니다. 게이트웨이가 질문을 보고 누구에게 넘길지 정해요.



아카이브는 유료 요금제에 포함돼요. Free에는 없습니다. Free에서 엔드포인트를 만들 때 대시보드에 이런 안내가 보여요.

> "무료 요금제에서는 아카이브 노드를 사용할 수 없습니다. 요청은 일반 노드가 처리하며, 노드가 보관 중인 데이터는 모두 응답합니다. 아카이브 노드에만 있는 데이터(과거 상태, 오래된 기록, 일부 트레이스)는 제공되지 않으며 노드가 자체 오류를 반환합니다."

- '옛' 질문에 따로 매기는 값은 없어요. 메서드에 대해 냅니다(25장).
- 아카이브의 깊이는 네트워크마다 달라요. 그 위에 중요한 것을 만들기 전에, 내 네트워크에서 가장 오래된 내 질문을 시험해 보세요.
- 공개 주소는 아카이브 질문을 받지 않아요.

직접 해보기: 공개 노드는 엄격하다

'지금'이 아니라, 겨우 열 블록 전, 약 2분 전의 블록 번호로 잔액을 물어볼게요.

```
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
```

이렇게 가까운 과거조차 공개 주소는 아카이브로 칩니다. 공개 주소에는 8장의 말로 물으세요. `latest`, `safe`, `finalized` 예요.

두 번째 시험: 노드는 버렸을 수도 있다

블록은 상태가 아니라 히스토리예요. 하지만 이것도 모두가 보관하지는 않습니다. 2025년 7월부터 Ethereum 노드 프로그램들은 스테이킹 전환 전에 나온 블록의 내용을 갖고 있지 않을 수 있어요(10장). 2015년 8월의 46,147번 블록(`0xb443`)을 달라고 해 볼게요.

```
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
```

"잘라낸 기록은 제공할 수 없다"는 뜻이에요. 노드는 솔직하게 "나한테 그건 없어"라고 말합니다. 유료 요금제에서는 일반 노드가 "나한테 없어"라고 하면, 게이트웨이가 아카이브 노드에게 다시 물어볼 수 있어요. 그 네트워크에 아카이브 노드가 있다면요.

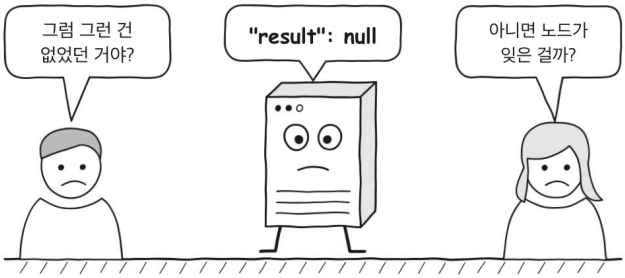
함정: "null이면 그런 일은 없었던 거지"

바로 그 블록에는 송금이 하나 있어요. 그 송금을 해시로 물어보고, 이어서 우리가 아무렇게나 지어낸 해시도 물어볼게요.

```
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}
```

대답이 똑같아요! 첫 번째 송금은 실제로 있었고, 두 번째는 존재한 적이 없습니다.



응답만으론 구별 불가

`null` 하나만 보고 결론 내리지 마세요. 오래된 기록이라면 더욱요. 어떤 때는 "없었다", 어떤 때는 "보관하지 않는다", 최근 송금이라면 "아직 블록에 없다"는 뜻입니다.

Solana 노드는 얼마나 기억할까

Solana는 1초에 블록을 여러 개 만들어서 기록이 산더미처럼 쌓여요. 최신 슬롯 번호와, 노드가 아직 가지고 있는 가장 오래된 블록을 물어볼게요.

```
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}
```

차이는 약 400,000슬롯이에요. 확인 당시에는 겨우 하루치였습니다. 1000번 블록은 이미 오래전에 치워졌어요.

```
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}
```

다시 회계사에게

'올해 초' 잔액은 지난해의 마지막 블록 직후의 잔액이에요(시간은 세계 표준시 UTC로 셉니다). 노드는 날짜로 블록 번호를 바로 알려 주지 않아요. 대신 블록은 순서대로 이어지니 이분 탐색을 할 수 있습니다. 가운데 블록을 집어 시각을 보고, 필요 없는 절반을 버리는 거예요.

2026년의 첫 블록을 찾아봅시다. 스크립트는 약 1분 걸려요.

```
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":'$2'}'
  sleep 2
}
WHEN=1767225600   # 2026년 1월 1일 00:00 UTC
low=20000000
high=$(( $(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 "응답 없음"; exit 1; fi
  if [ $(( t )) -lt $WHEN ]; then low=$(( mid + 1 )); else high=$mid; fi
  steps=$(( steps + 1 ))
done
echo "블록 $low, 질문 수: $steps"
```

```
블록 24136053, 질문 수: 23
```

수백만 블록 중에서 필요한 것을 23번의 질문으로 찾았어요(여러분의 횟수는 1 정도 다를 수 있어요). 따라서 2025년의 마지막 블록은 24,136,052(`0x1704974`)입니다. 그 블록의 잔액은 첫 번째 시험처럼 묻되, 이 번호를 쓰면 돼요. 공개 주소는 거절하니, 유료 요금제의 개인 엔드포인트가 필요합니다. 토큰이라면 `eth_getBalance` 대신, 컨트랙트에게 `balanceOf` 를 묻는 `eth_call` 을 써요.

긴 구간은 나눠서

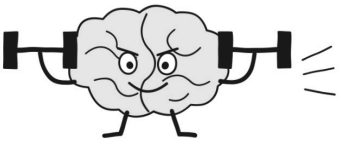
"1년 치 이벤트 전부"라는 요청은 노드가 아마 감당하지 못할 거예요. 응답이 너무 크니까요. 블록 구간을 여러 조각으로 잘라 차례로 묻고, 대답을 합칩니다.

이것은 대강의 모습이에요. `rpc` 는 23장의 `ask` 처럼 개인 엔드포인트에 묻는 여러분의 함수입니다.

```
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
}
```

통과되지 않으면 조각을 줄이세요. 조각 하나하나가 별도의 유료 호출이에요. 그리고 가게에 가장 싼 아카이브는 자기 것입니다. 결제는 보는 즉시 데이터베이스에 적으세요(26장).

생각해 보기



머리 좀 굴려 볼까!

1. 노드가 오래된 송금의 해시에 `null`을 돌려줬어요. 손님이 결제하지 않았다는 증거일까요?
2. 7월 1일의 USDC 잔액이 필요해요. Free 요금제로 충분할까요?
3. 5000블록의 이벤트를 1000블록씩 나눠 훑어야 해요. 호출은 몇 번일까요?

정답: 1) 아니요, 노드가 그 기록을 보관하지 않았을 수 있어요. 2) Free에는 아카이브가 없으니 기대하면 안 돼요. 3) 다섯 번.

핵심 정리

- 옛 잔액, 옛 이벤트, 트레이싱은 아카이브 노드의 일이에요.
- 아카이브는 유료 요금제에 포함되고, 엔드포인트 주소는 그대로예요.
- `null`이나 빈 대답이 아무 일도 없었다는 증거는 아니에요.

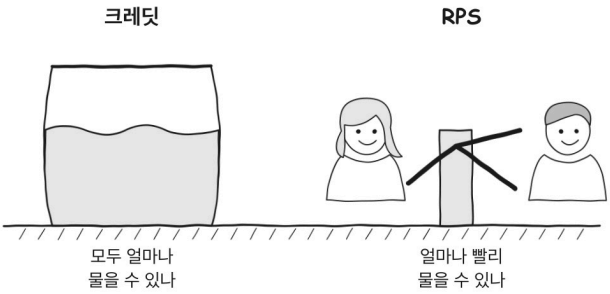
다음: 25장. 요금제, 호출 가격, 한도.

제25장. 요금제, 호출 가격, 한도

"그래서 한 달에 얼마죠?"

엔드포인트는 잘 돌아가요. 이제 아르티옴은 가게에 얼마가 들지 알고 싶습니다. 요금 페이지에는 중요한 숫자가 두 개 있어요. 크레딧과 초당 요청 수. 각각 무엇일까요?

알아보기: 서로 다른 두 척도



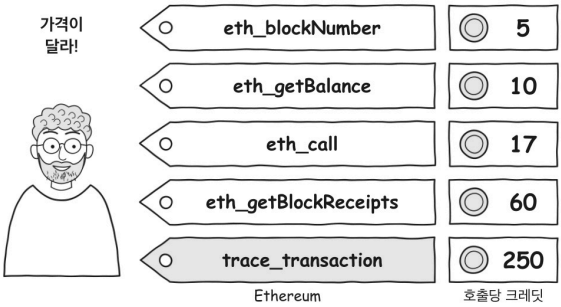
크레딧은 서비스 안의 '돈'이에요. 한 달 동안 모두 얼마나 물을 수 있는지입니다. RPS는 초당 요청 수, 즉 얼마나 빨리 물을 수 있는지예요. 둘 다 충분해야 합니다.

직접 해보기: 호출 가격 알아보기

질문마다 값이 달라요. 어떤 것은 노드가 메모리에서 바로 답하고, 어떤 것은 블록 하나를 통째로 뒤져야 합니다. 가격은 공개돼 있고, 가입 없이 물어볼 수 있어요. Ethereum 메서드 몇 개를 골라 볼게요(`python3` 가 필요해요).

```
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
```



- `trace_transaction` 하나가 블록 번호 질문 50개 값이에요.
- 네트워크마다 가격표가 따로 있어요. 주소의 `ethereum`을 `solana`나 `tron`으로 바꾸고 메서드 이름도 바꿔 보세요.
- 배치(11장)는 안에 든 호출들의 값을 합친 값이에요.
- WebSocket 구독도 유료예요. Ethereum에서 새 블록 구독을 여는 데 10크레딧, 전달되는 이벤트마다 2크레딧입니다.

대시보드에서는 가격이 '네트워크' 메뉴에 모여 있어요.

요금제

2026년 11월 27일 기준 자료예요. 최신 내용은 `rpcnode.dev/pricing`에 있습니다.

요금제	월 $	연 $	월 크레딧	RPS	엔드포인트
Free	0	—	1,500만	15	3
Starter	19	190	5,000만	40	5
Build	36	360	1억	75	12
Grow	99	990	3억	150	25
Scale	219	2,190	7억	300	50
Business	449	4,490	15억	500	100

요금제는 RPC만이 아니에요. 이런 것들도 들어 있습니다(26장과 27장).

요금제	감시 주소 수	결제: 동시 진행 / 월	트랜잭션 감시 수	아카이브와 전체 네트워크
Free	—	—	—	없음: Ethereum과 Solana만
Starter	15	10 / 200	15	있음
Build	50	25 / 1,000	40	있음
Grow	150	50 / 5,000	80	있음
Scale	400	100 / 20,000	150	있음
Business	1,000	250 / 100,000	400	있음

- 새 계정은 한 번, 카드 없이 무료로 Starter를 7일 동안 써 볼 수 있어요.
- 기성 요금제가 맞지 않으면, 요금 페이지에 '직접 구성하기'가 있어요.
- 요금제를 사려면 '구매'를 누르고 확인하세요. 15분짜리 청구서가 나오고, 암호화폐로 결제할 수 있어요.
- 기간이 끝났는데 새 결제가 없으면 계정은 Free로 넘어가요. 알림을 맞춰 두세요.

아르티옴의 가게를 계산해 보기

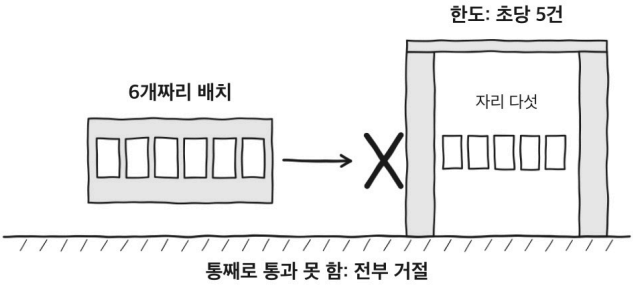
결제를 기다리는 주문이 20건이고, 각각 Ethereum 주소가 따로 있어요. 30일을 잡고, 주문이 늘 기다리고 있다고 칩니다. 넉넉하게 잡은 상한 추정이에요.

사이트가 묻는 방식	요청 수	크레딧	충분한 요금제
주소마다 10초에 한 번 잔액	5,184,000	51,840,000	Build
같은 것을 30초에 한 번	1,728,000	17,280,000	Starter
12초에 한 번 블록 번호와 `eth_getLogs` 하나	432,000	4,752,000	Free

세 번째 방식은 첫 번째보다 열 배 넘게 쌉니다. 질문 하나로 모든 주소를 한꺼번에 묻기 때문이에요. 다만 비교에는 단서가 있어요. 처음 두 방식은 ETH 입금을 알아채고, 세 번째는 토큰 송금을 알아챕니다(15장). 그리고 Free에는 네트워크가 Ethereum과 Solana 두 개뿐이에요.

초당 한도

RPS 계수기는 매초 0으로 돌아가고, RpcNode 문서에 따르면 엔드포인트마다 따로 있어요. 넘치는 요청은 HTTP 429와 `-32005` 코드를 받습니다. 호출 N개짜리 배치는 요청 N건이고, 통째로 통과하거나 통째로 거절돼요.



한도가 5인 공개 주소에서 확인해 봅시다. 질문 6개짜리 배치 하나면 충분해요.

```
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`은 "1초 뒤에 다시 보내라"는 뜻이에요. 공개 주소는 비싼 메서드도 아낍니다.

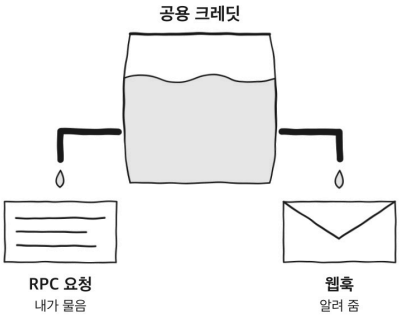
```
curl -s -w '  HTTP %{http_code}\n' https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBlockReceipts","params":["latest"]}'
```

```
{"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
```

무엇은 차감되고 무엇은 안 될까

무슨 일	크레딧
노드가 결과를 돌려줌	차감됨
노드가 내 요청에 대한 오류를 돌려줌, 예: `execution reverted`	차감됨: 이것도 대답이니까
초당 한도로 거절	차감 안 됨
네트워크 목록에 없는 메서드	차감 안 됨
어느 노드도 대답하지 않음	돌려줌
공개 주소로 보낸 요청	무료

비축분은 하나



웹훅, 즉 서비스가 여러분의 사이트로 직접 보내는 메시지(26장) 값도 같은 비축분에서 나갑니다. Ethereum 메인넷에서 감시 주소의 이벤트는 450크레딧, 결제는 750크레딧이에요. Ethereum 테스트넷에서는 절반 값입니다.

남은 양은 대시보드의 '사용량' 메뉴에서 보세요. 가장 자주 쓴 메서드도 거기 있어요.

크레딧이 떨어지면

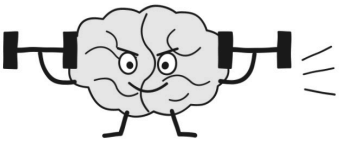
초과 사용 청구서는 없어요. 묶음 이상의 추가 요금은 없습니다. 하지만 서비스가 아무 일 없다는 듯 계속 돌아갈 거라고 기대하지는 마세요. 11장의 거절(`-32006`)을 프로그램에 대비해 두고, 남은 양을 미리 지켜보세요.

함정: 가격표만 보고 요금제 고르기

먼저 계산하세요. 몇 번 × 메서드 가격. 무거운 호출 몇 개가 가벼운 호출 수천 개보다 더 많이 먹을 수 있어요.

메서드 가격과 조건은 바뀔 수 있어요. 서비스 약관에 따르면, 법이 달리 요구하지 않는 한 요금제 결제는 환불되지 않고, 무중단 운영은 보장되지 않습니다. 유료 요금제에서 큰 장애가 나면 결제한 기간을 연장해 줘요. 결제하기 전에 약관을 읽으세요.

생각해 보기



머리 좀 굴려 볼까!

1. 사이트가 `eth_call`(17크레딧)을 한 달 30일 내내 10초마다 호출해요. 크레딧이 얼마나 들까요?
2. Free 요금제에서 호출 20개짜리 배치를 보냈어요. 어떻게 될까요?
3. 요청이 429를 받았어요. 크레딧이 차감됐을까요?

정답: 1) 259,200 × 17 = 4,406,400. 2) 통째로 거절돼요. 20은 15보다 커요. 3) 아니요.

핵심 정리

- 크레딧은 한 달의 양을, RPS는 속도를 제한해요.
- 메서드마다 값이 다르고, 배치는 호출 수만큼 계산돼요.
- 더 똑똑하게 묻는 것이 큰 요금제를 사는 것보다 이득일 때가 많아요.

다음: 26장. 주소 이벤트: 끝없이 묻는 대신 웹훅과 소켓.

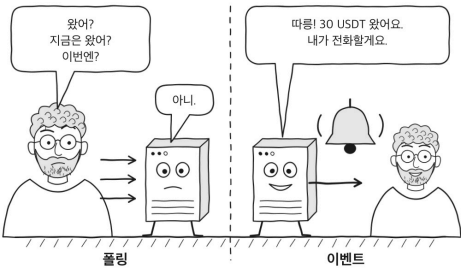
제26장. 주소 이벤트: 끝없이 묻는 대신 웹훅과 소켓

“주문 20건이면 초당 질문 20개?”

3장에서 계산했어요. 기다리는 주문 20건을 각각 1초에 한 번씩 확인하면, 벌써 Free 요금제가 허용하는 초당 15건을 넘습니다. 게다가 대답은 거의 다 똑같아요. “아직 안 왔어요.”

알아보기: 묻기와 전화 기다리기

폴링은 사이트가 몇 번이고 직접 “왔어?”라고 묻는 거예요. 이벤트는 일이 생기는 즉시 상대가 알려 주는 것입니다.



직접 해보기: 질문 하나로 모든 주소 묻기

우선 똑똑하게 물어봅시다. 주문마다 따로 물을 필요가 없어요. 15장의 `eth_getLogs` 메서드에는 받는 사람 목록을 한 번에 댈 수 있습니다.

```
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"}]}'
```

질문은 이거예요. “최신 블록에, 누구에게서든 이 세 주소 중 하나로 들어온 USDT 송금이 있어?” 주소는 64자가 되도록 왼쪽을 0으로 채웠어요. 대답은 이랬습니다.

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

빈 목록이에요. 이 블록에서는 아무도 결제하지 않았어요. 주소가 20개여도 요청은 여전히 하나입니다. 하지만 새 블록이 나올 때마다 매번 물어야 하고, 컨펌도 직접 세야 하며, ETH

자체의 송금은 따로 잡아야 해요(15장).

Address Watch: 대신 지켜봐 주기

Address Watch, 즉 주소 감시가 이 일을 대신 맡아요. RpcNode가 블록을 분석하다가, 트랜잭션이 여러분의 주소와 관련되면 여러분의 서버에 알려 줍니다.

주소는 대시보드에서 추가해요. '알림' → '주소' → '주소 추가'. 프로그램에서는 `POST /v1/address-watches` 요청에 `{"network_slug": "tron", "address": "내_주소"}` 본문을 담아 보내면 같은 일을 합니다.

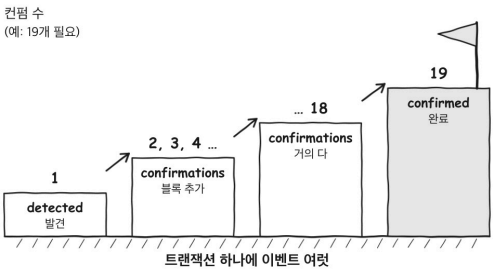
응답으로 감시 번호인 `watch_id`가 와요. 한 네트워크의 주소 하나가 요금제의 슬롯 하나를 차지합니다. 감시를 삭제하면 슬롯이 비지만, 추가와 삭제 횟수에도 한도가 있어요. Starter는 15슬롯이에요. Free의 메인넷에는 슬롯이 없고 '샌드박스'만 있어요. 테스트넷의 주소 몇 개입니다.

이벤트를 위해 블록을 분석하는 네트워크보다 더 많은 네트워크의 엔드포인트를 받을 수 있어요. 2026년 11월 27일 기준 분석 대상은 Ethereum, Bitcoin, Solana, TRON, BNB Smart Chain, Base, Arbitrum One, TON, Litecoin, Dogecoin, Bitcoin Cash, Dash, Mantle, Monad이고, 테스트넷은 TRON Nile뿐입니다.

돈이 들어오면 `payment`, 나가면 `payout`, 그 밖의 모든 것(예를 들어 approve 허용)은 `address` 종류의 이벤트예요. 새 블록은 `block` 종류이고, `tx`와 `payment_order`는 27장에서 다룹니다.

트랜잭션 하나에 벨은 여러 번

송금은 한 번만 알리는 게 아니라, '굳어 가는' 과정에 따라 알려요(8장). 정확히 무슨 일이 있었는지는 `event` 필드가 말해 줍니다.



트랜잭션 하나에 이벤트 여럿

event	뜻	가게가 할 일
`detected`	블록에서 트랜잭션이 보임	"송금이 확인되고 있어요" 표시
`confirmations`	컨펌 하나 추가	숫자 갱신
`confirmed`	컨펌 충분, 성공	돈이 도착한 것으로 처리
`failed`	컨펌 충분, 트랜잭션 실패	발송하지 않기

컨펌을 몇 개 기다릴지는 '설정' → '컨펌 수'에서 정해요. 정하지 않으면 네트워크의 기본값이 적용됩니다.

Bitcoin, Litecoin, Dash에서는 `pending`이 가장 먼저 올 수 있어요. 송금이 아직 멤풀(7장)에 있다는 뜻입니다. 이때 종류는 `payment`가 아니라 언제나 `address`이고, 아직 돈이 아니에요. 이런 송금은 교체될 수 있습니다.

웹훅: 내 우편함에 오는 편지

웹훅은 서비스가 인터넷상의 여러분 주소로 직접 보내는 요청이에요. 주소와 네트워크별 구독은 '알림' → '전송 설정'에서 지정합니다. 주소는 인터넷에서 접근할 수 있어야 해요. `localhost`는 안 됩니다.

아무것도 고르지 않고 웹훅을 켜면, 모든 네트워크의 입금과 출금이 오고 블록은 오지 않아요.

문서 예제의 일부예요. TRON의 30 USDT에 관한 첫 번째 벨입니다.

```
{ "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 }
```

금액은 반올림되지 않도록 문자열로 와요. `amount_raw`는 소수점 없는 같은 값입니다(9장). 서비스 목록에 없는 토큰이라면 `amount_raw`만 있어요. 헤더에는 `X-RpcNode-Delivery`(전송 번호), `X-RpcNode-Event`(종류), `X-RpcNode-Signature`(서명)가 옵니다.

"정말 당신들 맞아요?"

웹훅 주소는 열려 있어요. 누구든 두드려서 "돈 받으셨어요"라고 쓸 수 있습니다. 그래서 모든 전송에 서명이 붙어요. 시각, 점, 본문을 이어서 웹훅 시크릿으로 지문을 계산합니다. 이 방식을 HMAC-SHA256이라고 해요. 시크릿은 '시크릿 및 IP' 항목에 있어요. API 키의 시크릿과는 다릅니다.

직접 해보기: 직접 서명해 보기

여기의 시크릿과 본문은 지어낸 거예요.

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

```
e17efb57778b0e0c758a9bf2c5705049c4c383d2af3a2bd2783c8b665c151cde
```

본문이나 시크릿의 글자 하나만 바꿔도 완전히 다른 문자열이 나와요. 여러분의 서버는 같은 계산을 해서 `t=…,v1=…` 모양의 헤더와 비교합니다.

Node.js로 만든 수신기의 대강의 모습이에요. 직접 만든 요청으로는 시험했지만, 진짜 전송으로는 시험하지 못했어요. 그러려면 대시보드 계정이 필요합니다. 300초 검사는 어제의 전송을 다시 보내는 것을 막아 줘요.

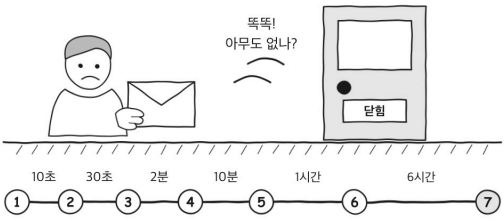
```
import { createServer } from 'node:http'
import { createHmac, timingSafeEqual } from 'node:crypto'

const SECRET = process.env.RPCNODE_WEBHOOK_SECRET
const seen = new Set()      // 실제 가게라면 데이터베이스
createServer(async (req, res) => {
  const parts = []
  for await (const part of req) parts.push(part)
  const body = Buffer.concat(parts)   // 도착한 그대로의 바이트
  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())  // 여기에 할 일을 작성
  }
  res.writeHead(200).end()
}).listen(3000)
```

집에 없었다면

서버가 10초 안에 200번대 코드로 대답하면 전송 성공이에요. 그렇지 않으면 다시 보내고, 일곱 번 실패하면 포기합니다. 그러므로 이렇게 하세요.

- 같은 이벤트가 두 번 올 수 있어요. 전송 번호가 같으니, 그것으로 걸러 내세요.
- 순서는 약속되지 않아요. `confirmations`가 `detected`를 앞지를 수 있습니다.
- 오래 걸리는 일은 응답한 뒤에 하세요.



간격은 점점 길게, 모두 일곱 번, 약 일곱 시간

내 주소는 시험 전송으로 미리 확인하세요. 무료예요. 대시보드의 전송 기록은 하루 동안 보관됩니다.

소켓: 열어 둔 선

두 번째 방법은 WebSocket이에요. 여러분의 프로그램이 직접 연결을 열고 듣기 때문에, 인터넷상의 내 주소가 필요 없습니다. 11장의 `eth_subscribe` 구독과는 달라요. 거기서는 노드가 네트워크 이야기를 하고, 여기서는 서비스가 여러분의 주소와 주문 이야기를 합니다. 본문은 웹훅과 같지만, 자리를 비운 동안 놓친 것은 다시 보내 주지 않아요.

프로토콜은 Pusher와 호환되고(`pusher-js` 라이브러리를 쓸 수 있어요), 키는 API 키의 번호예요. 소켓은 'WebSocket으로 전송' 스위치로 따로 켭니다. 두 방법을 모두 켜면 소식마다 두 번 전달되고 두 번 요금이 나가요.

비용은 얼마일까

이벤트는 요청과 같은 크레딧을 써요. TRON이나 Ethereum 메인넷의 주소 이벤트는 2026년 11월 27일 기준 450크레딧이고, 트랜잭션당 한 번만 냅니다. 컨펌과 최종 결과는 무료예요. 전송에 대해 내는 것이라, 여러분의 서버가 나중에 받지 못했어도 냅니다.

Ethereum에서의 30일 한 달을 어림해 봅시다. 표의 요청은 하나에 17크레딧입니다.

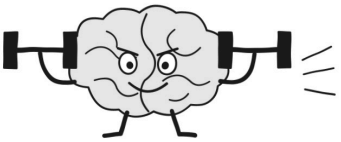
결제를 아는 방법	요청 또는 이벤트 수	크레딧
주문 20건, 10초마다 `eth_call` (13장)	5,184,000	88,128,000
블록마다 `eth_getLogs` 하나, 12초에 한 번	216,000	3,672,000
Address Watch, 한 달 송금 200건	200	90,000

Starter에는 5,000만 크레딧이 있어요. 첫 번째 방법은 여기에 다 들어가지 않습니다. 이것은 예일 뿐이에요. 송금이 많을수록 이벤트 비용이 커지고, 남이 내 주소로 보낸 자잘한 송금도 계산에 들어갑니다.

함정: "이벤트가 왔으니 발송"

- 모든 이벤트가 결제는 아니에요. `detected` 는 아직 최종이 아니고, `payout` 은 내 지출이에요.
- 토큰은 이름이 아니라 `token` 필드의 컨트랙트 주소로 알아보세요(9장).
- 특별한 송금은 미리 시험해 보세요. 예를 들어 컨트랙트 안에서 보낸 ETH(15장).
- 주소 이벤트는 아카이브가 아니에요. API로는 하루 동안만 볼 수 있어요.

생각해 보기



1. 가게 서버를 한 시간 껐어요. 웹훅 전송은 어떻게 되고, 소켓으로 가는 소식은 어떻게 될까요?
2. `X-RpcNode-Delivery`가 똑같은 전송이 두 번 왔어요. 돈을 몇 번 입금 처리해야 할까요?
3. 한 달에 감시 주소로 TRON 송금이 1000건 왔고, 웹훅으로만 받아요. 크레딧은 얼마일까요?

정답: 1) 웹훅은 일곱 번의 시도가 끝날 때까지 다시 보내고, 소켓은 놓친 것을 다시 보내지 않아요. 2) 한 번. 3) 450,000.

핵심 정리

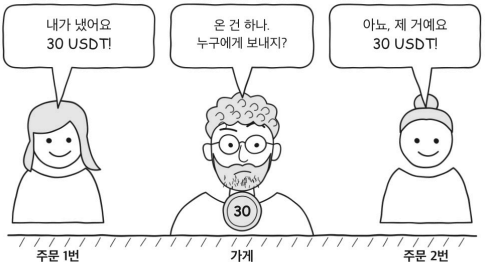
- 끝없이 묻지 마세요. 송금을 알려 주게 하세요.
- 서명을 확인하고, 중복을 걸러 내고, 빨리 대답하세요.
- 돈이 도착한 것은 첫 번째 벨이 아니라 `confirmed`에서예요.

다음: 27장. 결제와 트랜잭션 추적.

제27장. 결제와 트랜잭션 추적

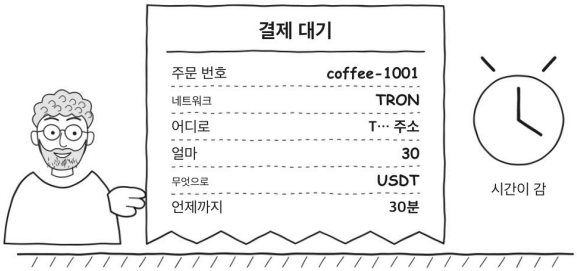
"돈이 왔어요. 그런데 누구 거죠?"

아르티옴이 이벤트를 받았어요. 주소에 30 USDT가 들어왔대요. 주문 목록을 열어 보니 30짜리가 두 건이에요. 이벤트는 주소와 금액은 알지만, 손님의 장바구니는 전혀 모릅니다.



알아보기: 결제는 곧 '기다림'

결제(대시보드의 '결제' 메뉴)는 미리 적어 둔 기대예요. 어떤 주문이 어떤 주소로 얼마를 몇 시까지 기다리는지요. 맞는 송금이 오면 서비스가 알아서 주문 번호를 알려 줍니다.



주소는 여러분의 것을 넣어요. 돈은 손님에게서 여러분에게 곧바로 가고, 서비스에는 시드 문구가 필요 없습니다(6장).

결제는 '새 결제' 버튼이나 `POST /v1/payments` 요청으로 만들어요.

필드	내용
`order_id`	내 주문 번호, 중복 불가
`network_slug`, `address`	네트워크와 거기의 내 주소
`amount`	문자열로 쓴 금액: `"30"`, `"12.5"`
`token_contract`	토큰 컨트랙트. 없으면 네트워크의 코인을 기다림
`ttl_seconds`	초 단위 기한. 없으면 1시간, 최대 30일

직접 해보기: 키 없이 두드려 보기

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

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

401 코드의 응답이에요. "그런데 누구세요?"라는 뜻이죠. 이 API의 모든 요청에는 API 키의 시크릿으로 서명해요('API 키' 메뉴). 네 줄에 서명합니다. 시각, 메서드, 경로, 본문의 지문이에요.

셸 언어로 그린 대강의 모습이에요. 문법과 서명 계산은 확인했지만 보내 보지는 않았어요. 개인 키가 필요하니까요.

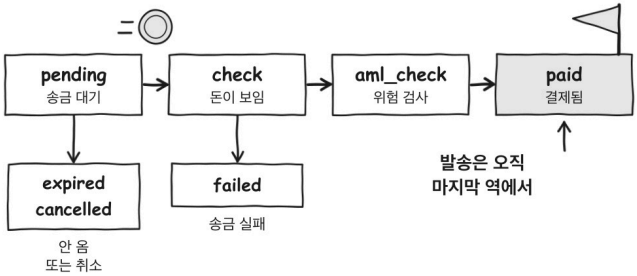
```
BODY='{"order_id":"coffee-1001","network_slug":"tron",
"address":"내_주소","amount":"30","ttl_seconds":1800,
"token_contract":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t"}'
TS=$(date +%s)
HASH=$(printf '%s' "$BODY" | openssl dgst -sha256 | awk '{print $NF}')
SIG=$(printf '%s\nPOST\n/v1/payments\n%s' "$TS" "$HASH"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

응답으로 `payment_id`, `pending` 상태, 만료 시각 `expires_at` 이 와요. 같은 `order_id` 와 같은 네트워크, 주소, 토큰, 금액으로 요청을 다시 보내면 같은 결제가 돌아옵니다. 다른 값이면 `payment.order_exists` 오류예요.

결제가 가는 길

단계마다 `payment_order` 이벤트가 와요. 웹훅이나 소켓으로요(26장).



event	무슨 일
`check`	돈이 보였고, 컨펌을 기다림
`confirmations`	컨펌이 늘어남
`aml_check`	컨펌 충분, 위험 검사 중
`paid`	결제 완료: 발송해도 됨
`expired`, `cancelled`	기한이 지났거나, 대기 중인 결제를 직접 취소함
`failed`	네트워크에서 트랜잭션 실패(이 상태에 모든 것을 걸지 마세요. 결제가 그냥 만료될 수도 있어요)

AML은 들어온 돈이 나쁜 이력과 연결돼 있지 않은지 보는 검사예요. 검사가 있는 곳에서는 `paid` 이벤트의 `aml` 객체에 0부터 100까지의 점수 `score`(높을수록 위험)와 등급 `risk_level`이 담깁니다. 별도 요금은 없어요. 모든 네트워크에 있는 것은 아니고, 없는 곳에서는 결제가 바로 `paid`가 됩니다.

이것은 참고 정보이지, 판결도 법률 의견도 아니에요. 위험한 송금을 어떻게 할지는 여러분이 정합니다. 결제가 `aml_check`에서 멈췄다면 아직 결제된 것이 아니니, 지원팀에 문의하세요.

문서 예제의 일부예요.

```
{ "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" } }
```

작은 글씨

- 모자란 금액은 인정되지 않아요. 결제는 계속 기다리다 만료됩니다. 문서에는 `underpaid` 상태가 있지만, 가게를 거기에 기대어 만들지 마세요. 부족 결제는 주소 이벤트로

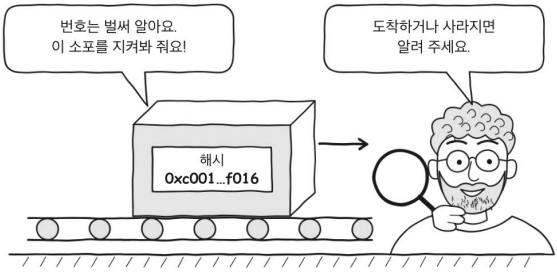
알아채세요.

- 넘치는 금액은 인정돼요. 실제로 얼마가 왔는지는 `received_amount_raw`가 알려 줍니다.
- 늦은 송금은 결제를 완료하지 못해요. 제때 보인 송금은 기한이 지나도 차분히 컨펌을 모읍니다.
- 주문마다 주소를 따로 두세요. 그러면 '30짜리' 두 건이 헷갈리지 않아요(28장).
- 주소 슬롯은 저절로 비지 않아요. `paid`나 `expired` 뒤에는 감시를 삭제하세요. 결제를 취소하면, 그 결제만을 위해 추가한 주소는 저절로 해제됩니다.

결제는 유료 요금제에 있어요. Starter는 동시 진행 10건, 월 200건입니다. TRON이나 Ethereum 메인넷의 결제 한 건은 750크레딧이고, 돈이 보였을 때만 차감돼요. Free에는 샌드박스가 있어요. 테스트넷에서 동시 진행 결제 3건이고, 지금은 TRON Nile입니다. 거기의 위험 검사는 진짜가 아니며 `"sample": true`로 표시돼요.

트랜잭션 추적

거꾸로인 경우도 있어요. 해시를 이미 가지고 있는 거죠. 아르티옴이 손님에게 환불했거나 공급업체에게 대금을 보내고, 도착했는지 알고 싶어요.



'알림' → '트랜잭션' → '해시 추가', 또는 `POST /v1/tx-watches`에 `{"network_slug": "tron", "tx_hash": "해시"}` 본문을 담아 보내세요. 해시는 네트워크에 나타나기 전에 추가해도 돼요.

`tx` 이벤트가 옵니다. `detected`, 이어서 `confirmations`, 마지막에 `confirmed` 또는 `failed`예요. 해시를 끝내 찾지 못하면 `cancelled`입니다. 이 네트워크에 입금이나 출금 알림이 켜져 있어야 와요.

진행 중인 감시는 슬롯 하나를 차지하고, 끝나면 비워 줍니다. Starter는 15슬롯이고, Free에는 이 기능이 없어요. 트랜잭션은 한 번만 요금을 냅니다. TRON과 Ethereum에서는 150크레딧이에요.

직접 해보기: 컨펌을 직접 세기

이런 감시의 속에서 벌어지는 일은 이거예요. 15장의 USDT 송금을 가져옵시다.

```
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"}
```

상태 `0x1`은 성공이에요. 송금이 든 블록이 첫 번째 컨펌이니, `echo $((0x19464c5 - 0x194646a + 1))`은 92를 줍니다. 여러분은 더 큰 수가 나올 거예요.

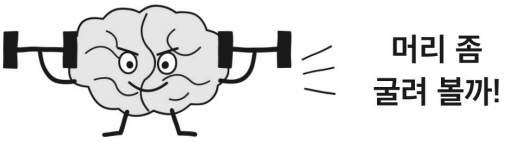
함정: 같은 돈에 벨이 두 번

결제를 만들면 주소에 감시가 걸려요. 그래서 송금 하나에 대해 `payment`(주소 이벤트)와 `payment_order`(결제 이벤트)가 둘 다 올 수 있습니다. 요금은 각각의 가격대로 나가요.

> 원칙: 주문은 `event = paid`인 `payment_order`만이 마감해요. 그리고 주문 번호, `payment_id`, 토큰, 금액을 대조하세요.

그리고 결제에 대해 끝없이 묻지 마세요. API에는 자체 한도가 있어요. 계정당 분당 100건입니다. 장애로 멈췄다가 돌아왔을 때 대조용으로 물어보세요. `GET /v1/payments/orders/{orderId}`입니다.

생각해 보기



1. 30 USDT 결제인데 29.5가 왔어요. 결제는 어떻게 될까요?
2. 기한이 지나고 1분 뒤에야 송금이 보였어요. 돈은 어디 있고, 상태는 어떨까요?
3. `check` 이벤트가 왔어요. 커피를 보내도 될까요?

정답: 1) 결제 완료가 안 되고 기한에 만료돼요. 2) 돈은 내 주소에 있고, 결제는 `expired`로 남아요. 손으로 처리해야 해요. 3) 아니요, `paid`를 기다리세요.

핵심 정리

- 결제는 송금을 주문과 이어 주고, 해시 감시는 트랜잭션의 운명을 알려 줘요.
- 발송은 첫 번째 벨이 아니라 `paid`를 보고 해요.
- 부족 결제, 지각, 차 있는 슬롯은 여러분이 챙길 일이에요.

파트 5 끝. 아르티옴은 엔드포인트를 받을 줄 알고, 요청의 길과 요금제를 이해하며, 묻지 않고도 돈 소식을 알게 됐어요.

다음: 파트 6, 28장. 가게 조립하기: 청구서, 주소, 결제 기다리기, 컨펌.

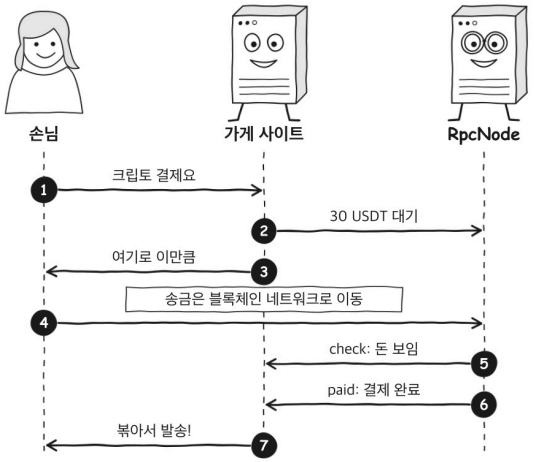
제28장. 가게 조립하기: 청구서, 주소, 결제 기다리기, 컨펌

“하나씩은 다 알겠어요. 합치면요?”

아르티옴은 주소, 컨펌, 웹훅, 결제를 알아요. 그리고 이제 손님이 ‘암호화폐로 결제’를 누릅니다. 사이트는 그다음에 한 단계씩 무엇을 해야 할까요?

알아보기: 누가 참여하나

참가자는 넷이에요. 지갑을 든 손님. 주문 데이터베이스가 있는 가게 사이트. 네트워크를 지켜보는 RpcNode. 그리고 키가 들어 있는 아르티옴의 지갑. 사이트와 서비스의 대화에 지갑은 끼지 않아요.

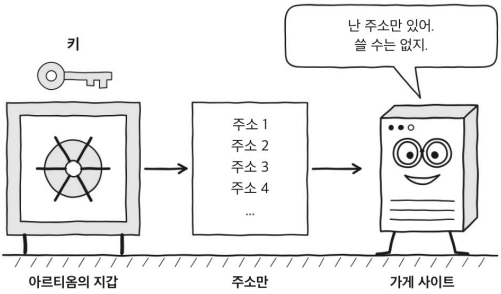


4단계를 보세요. 돈은 사이트로도, RpcNode로도 가지 않아요. 아르티옴의 주소에 곧바로 쌓이고, RpcNode는 블록에서 알아챌 뿐입니다.

서버에 키 없이 주소를 구하는 법

6장에서 약속했죠. 주문마다 주소를 따로 둔다고요. 거기서 또 말했어요. 가게 서버에 개인 키가 있을 자리는 없다고. 둘을 어떻게 함께 할까요?

주소는 키와 따로 미리 준비할 수 있어요.



방법은 두 가지예요.

	주소 목록	확장 공개 키
서버에 두는 것	주문마다 하나씩 쓸, 만들어 둔 주소들	키 하나(보통 `xpub`로 시작). 프로그램이 이것으로 주소를 직접 계산
쓸 수 있는 곳	어느 네트워크든	Bitcoin, Ethereum, TRON과 친척들. Solana와 TON에서는 이렇게 안 됨
서버가 해킹되면	목록의 주소가 보임	내 모든 주소와 거기 들어온 모든 돈이 보임
번거로움	제때 채워 넣기	한 번만 설정

이런 키로는 돈을 쓸 수 없어요. 하지만 BIP-32 표준은 경고합니다. 이 키와, 같은 가지의 개인 키 단 하나가 만나면 그 가지의 모든 개인 키를 계산해 낼 수 있어요. 그러니 이런 지갑의 키를 하나씩 내보내지 마세요. 초보자는 목록으로 시작하는 편이 쉬워요.

목록은 키가 인터넷의 서버에 놓여 있지 않은 지갑에서 가져오세요. 하드웨어 지갑이 편해요. 키를 자기 안에 보관하는 별도의 기기입니다. 올리기 전에 주소 몇 개를 기기 화면과 대조하세요. 그리고 TON에서는 손님에게 `UQ` 형태의 주소를 보여 주세요(19장).

직접 해보기: 데이터베이스는 헷갈리게 두지 않는다

주소 나눠 주기는 데이터베이스의 일이에요. 여기 장난감이 하나 있습니다. 빈 주소 둘과 주문 셋이에요. `sqlite3` 프로그램이 필요한데, 많은 시스템에 이미 들어 있어요.

```
sqlite3 :memory: <<'SQL'
CREATE TABLE addresses(address TEXT PRIMARY KEY, order_id TEXT UNIQUE);
INSERT INTO addresses(address) VALUES ('주소-1'), ('주소-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 '세 번째에는 주소가 모자라요' WHERE changes() = 0;

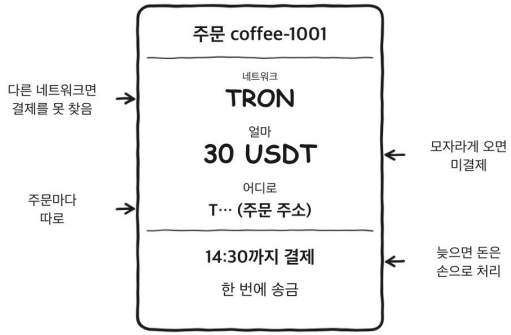
UPDATE addresses SET order_id = 'coffee-1001' WHERE address = '주소-2';
SQL
```

```
coffee-1001|주소-1
coffee-1002|주소-2
세 번째에는 주소가 모자라요
Runtime error near line 15: UNIQUE constraint failed: addresses.order_id (19)
```

세 번째 청구서는 낼 수가 없어요. 목록을 채울 때입니다. 그리고 마지막 명령은 1001번 주문에 두 번째 주소를 주려 했는데, 데이터베이스가 거절했어요. `PRIMARY KEY`와 `UNIQUE`라는 말이 지켜 주는 거예요. 주소는 한 번만 적히고, 주문에 주소는 하나를 넘지 않습니다.

그리고 옛 주소를 두 바퀴째 돌려쓰지 마세요. 지난 주문의 늦은 송금이 새 주문을 결제해 버립니다.

청구서 내기



손님에게 필요한 것은 다섯 가지예요. 네트워크, 코인, 정확한 금액, 주소, 기한. 그리고 부탁 하나. 전부를 한 번에 송금해 달라는 것이에요. 반쪽 두 개는 합쳐지지 않으니까요. 미리 알려 주세요. 지갑이나 거래소가 금액에서 수수료를 떼면 덜 도착한다고요.

루블이나 달러 가격은 주문하는 순간의 코인으로 환산하고, 기한이 끝날 때까지 그 환율을 지켜요. 그래서 기한은 짧게 둡니다. 예를 들어 30분.

이제 27장의 결제가 필요해요. 아래는 Python으로 그린 대강의 모습입니다. 여기의 `rpcnode` 함수는 서명한 요청을 보내요. 27장을 따라 직접 쓰거나, RpcNode의 공식 클라이언트를 쓰세요(Python, TypeScript, PHP, Go, Kotlin용이 있어요). 데이터베이스는 아까처럼 테이블 두 개짜리입니다.

```
USDT = 'TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t'    # TRON의 USDT 컨트랙트
# addresses(address, order_id)
# orders(id, payment_id, amount_raw, status — 처음에는 'waiting')

def take_address(db, order_id):
    with db:                                    # 데이터베이스 트랜잭션 하나
        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]                      # 다시 부르면 같은 것
        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('주소가 바닥났어요')
        return free[0][0]
```

```
def checkout(db, rpcnode, order_id, amount):   # amount는 문자열
    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        # 손님에게: address, amount, expires_at
```

여기서 핵심은 순서예요. 서비스에 말을 걸기 전에 주소를 주문에 묶어 둡니다. 연결이 끊겨도, 다시 호출하면 같은 주소와 주문 번호를 쓰고, 서비스는 같은 결제를 돌려줘요. 지어낸 `rpcnode` 함수로 확인했고, 진짜 서비스로는 이 코드를 돌려 보지 않았습니다.

결제는 장바구니마다가 아니라, 손님이 암호화폐 결제를 골랐을 때 만드세요. 요금제의 월 한도는 만들어진 결제를 세요. 결제되지 않은 것도요.

RpcNode는 주소, 코인, 금액으로 결제를 알아봐요. 19장의 코멘트와 18장의 `reference` 표식은 여기에 쓰이지 않습니다. 결제에는 그런 필드가 없어요. 20장의 XRP Ledger와 Stellar는 결제가 가능한 네트워크 목록에 들어 있지 않아요(26장).

결제 기다리기

주문 페이지는 몇 초에 한 번 내 서버에게 묻고, 서버는 자기 데이터베이스에서 답해요. 데이터베이스는 웹훅이 갱신합니다. 브라우저에서 RpcNode에 직접 물으면 안 돼요. 키의

시크릿이 손님에게 넘어가고, API 한도도 계정당 분당 100건이니까요.

상태	손님이 보는 것	가게가 하는 일
`pending`	“14:30까지 송금을 기다립니다”	아무것도 안 함
`check`	“송금이 확인됩니다. 컨펌 19개 중 3개”	발송하지 않음
`aml_check`	“송금을 검사하고 있습니다”	기다림
`paid`	“결제 완료, 주문을 준비합니다”	발송함
`expired`	“기한이 지났습니다. 이 청구서로 송금하지 마세요”	청구서를 닫음
`failed`	“송금이 처리되지 않았습니다”	다시 시작하자고 안내
`cancelled`	“청구서가 취소됐습니다”	청구서를 닫음

송금이 블록에 들어가기 전까지 결제는 말이 없어요. 그러니 “결제 안 됨”이 아니라 “기다리는 중”이라고 쓰세요.

컨펌은 몇 개 기다릴까

‘설정’ → ‘컨펌 수’에서 네트워크마다 따로 정해요. 정하지 않으면 네트워크의 기본값이 적용됩니다. 필요한 수는 결제의 `required_confirmations` 필드에서 볼 수 있어요.

기본값보다 적게 잡는 것은 위험해요. 갓 나온 블록은 가끔 바뀌니까요(8장). 그리고 이 숫자는 주문이 아니라 네트워크당 하나예요. 커피 한 잔보다 원두 한 자루를 더 오래 기다리고 싶다면, `paid` 뒤에 더 기다렸다가 27장처럼 직접 확인하세요.

자리를 비우세요

자리는 두 종류예요. 결제 슬롯은 결제가 `paid`, `expired`, `failed`, `cancelled`가 되면 저절로 비어요. 주소 슬롯은 저절로 비지 않습니다(취소는 예외, 27장).

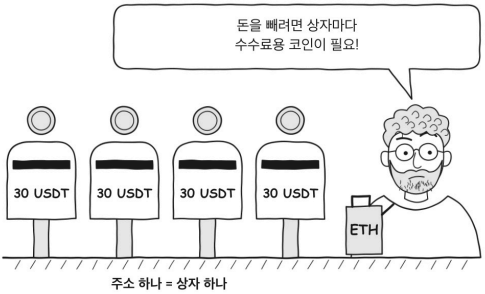
Starter는 동시 진행 결제 10건, 감시 주소 15개, 월 결제 200건이에요. 열한 번째 동시 결제는 `payment.slot_limit` 거절을 받습니다. 주소 자리가 다 차면 `watch.address_limit`이 와요.

닫힌 주문은 송금이 늦었을 수도 있으니 잠시 그대로 두세요. 그다음 감시를 치웁니다. `GET /v1/address-watches?network_slug=tron&address=…` 요청으로 `watch_id`를 찾아 `DELETE /v1/address-watches/{watch_id}`를 보내세요. 그 뒤로는 그 주소의 소식이 오지 않아요. 돈은 여전히 그 주소에 도착할 수 있는데도요.

주소를 추가하고 삭제하는 횟수에도 하루, 일주일, 한 달 단위의 제한이 있고, 슬롯이 많을수록 여유가 큽니다. 주문마다 새 주소를 쓰면 이 여유가 닳아요. 세일하는 날에는 빈 슬롯이 있어도 `watch.quota`를 받을 수 있어요. 주문이 많을 것 같으면 한계를 미리 알아 두세요.

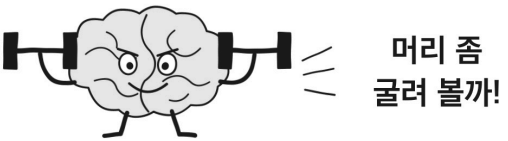
함정: 주소 백 개에 흩어진 돈

주문마다 주소를 두면 장부 정리는 편하지만, 돈이 따로따로 놓여요. 어떤 주소의 USDT를 옮기려면 바로 그 주소에 수수료용 코인이 있어야 합니다(9장과 17장). Ethereum은 ETH, TRON은 TRX나 에너지예요. 주문 백 건이면 송금도 따로 백 번입니다. Bitcoin에서도 작은 '조각'이 많으면 보내는 값이 비싸져요(14장). 소액 주문용 네트워크를 고르기 전에 이것을 계산하세요.



하나 더. BIP-44 표준 지갑은 빈 주소가 연달아 20개 나오면 더 찾지 않아요. 버려진 주문이 많으면, 멀리 있는 주소의 결제를 지갑이 보여 주지 않을 수 있습니다. 돈은 무사하지만, 지갑이 더 멀리까지 찾을 수 있는지 확인하세요.

생각해 보기



1. 손님이 결제 페이지를 새로고침했어요. 새 주소와 새 결제가 필요할까요?
2. Starter를 쓰는 아르티옴에게 이미 동시 결제가 10건 있어요. 열한 번째 손님이 오면 어떻게 될까요?
3. 결제 페이지가 브라우저에서 RpcNode에 직접 물어요. 문제 두 가지를 말해 보세요.

정답: 1) 아니요, 같은 주문 번호와 같은 주소예요. 2) `payment.slot_limit` 거절. 잠시 기다려 달라고 하거나 다른 결제 방법을 안내해요. 3) 키의 시크릿이 손님에게 보이고, 요청 한도가 금방 바닥나요.

핵심 정리

- 키는 지갑에 남고 사이트에는 주소만 둬요. 주문 하나에 주소 하나, 재사용 없이.
- 손님은 내 사이트에 묻고, 사이트는 웹훅으로 소식을 알아요.
- 슬롯과 돈을 모으는 비용은 미리 계산하세요.

다음: 29장. 모든 것이 꼬일 때: 오류, 재시도, 멈춘 트랜잭션, 이중 입금 처리.

제29장. 모든 것이 꼬일 때: 오류, 재시도, 멈춘 트랜잭션, 이중 입금 처리

"커피가 두 번 나갔어요!"

아침에 아르티옴이 보니, 주문 하나에 봉투 두 개가 발송됐어요. 다른 손님은 이렇게 써요. "돈을 냈는데 거기는 계속 '대기 중'이에요." 이상한 일이 아니에요. 맑은 날씨만 생각하고 만든 프로그램에서는 이런 일이 생깁니다.

알아보기: 세 가지 말썽

말썽	예	처방
내 요청에 대답이 없음	결제를 만드는 중에 연결이 끊김	해 없이 다시 보내기
소식이 이상하게 옴	두 번, 뒤죽박죽 순서로, 또는 영영 안 옴	데이터베이스의 자물쇠와 대조
송금이 이상하게 감	멈춤, 지각, 모자란 금액	규칙과 손으로 처리

API 요청은 언제 다시 보낼까

쉬는 시간을 늘려 가는 방법은 23장에서 살펴봤어요. 하지만 겁 없이 다시 보낼 수 있는 것은 멱등한 것뿐입니다. 몇 번을 해도 결과가 한 번 한 것과 같은 것이요. 결제 만들기는 `order_id` 덕분에 그렇습니다. 같은 번호와 같은 데이터로 다시 보내세요. 다시 보낼 때마다 서명은 새로 해야 해요. 서명에 시각이 들어 있으니까요.

RpcNode의 공식 클라이언트는 끊김이나 500번대 뒤에 `POST`를 알아서 다시 보내지 않아요. 그러니 결제 만들기를 다시 시도하는 것은 여러분 몫입니다.

API의 여러 응답에 대처하는 법이에요.

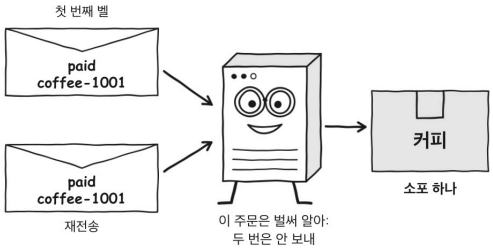
온 것	할 일
아무것도 없음, 502, 503, 504	쉬어 가며 다시 보내기
`Retry-After` 헤더가 붙은 429	적힌 만큼 기다리기
429 `watch.quota` (28장)	바로 다시 보내도 소용없음: 여유분이 바닥남
400	요청의 오류: `errors`에 필드 이름이 있음
401	키, 서명, 서버 시계 확인
403	요금제, 슬롯 한도, 또는 IP 허용 목록
409 `payment.order_exists`	이 번호는 다른 데이터의 결제가 이미 씀: 번호를 대충 바꾸지 말고 원인 파악

결제가 만료됐는데 손님이 다시 해 보고 싶어 하는 경우는 달라요. 옛 `order_id`는 똑같이 만료된 결제를 돌려줍니다. 새 번호가 필요해요. 예를 들어 `coffee-1001-2`.

소식 하나가 두 번

쌍둥이는 어디서나 생겨요.

- 10초 안에 대답하지 않아서 웹훅이 다시 왔어요.
- 웹훅과 소켓이 둘 다 켜져 있어요.
- 같은 돈에 대해 `payment`와 `payment_order`가 왔어요(27장).
- 웹훅과 정해진 시간의 점검이 동시에 돌았어요.
- 직원이 손으로 '발송'을 눌렀어요.



전송 번호(26장)는 같은 전송의 반복을 걸러 내요. 하지만 다른 길로 온, 같은 돈에 관한 두 번째 소식은 막지 못합니다. 가장 중요한 자물쇠는 주문의 상태예요. '대기'에서 '결제됨'으로는 딱 한 번만 넘어갈 수 있어야 합니다.

직접 해보기: 웹훅 하나를 두 번

자물쇠 전체는 이렇게 생겼어요. 소식은 우리가 지어냈습니다. 두 조각을 `once.py` 파일 하나에 저장하고 `python3 once.py`로 실행하세요.

```
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 '아직 결제가 아님'
    with db:                # 안의 모든 것이 데이터베이스 트랜잭션 하나
        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 '이미 처리했거나 우리 것이 아님'
        db.execute('INSERT INTO to_ship VALUES (?)', (n['order_id'],))
    return '발송'
```

이어서 소식 네 개를 연달아 넣고 프로그램의 대답을 봅니다.

```
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))                      # 같은 웹훅 한 번 더
print(4, on_news({**paid, 'event': 'check'}))  # 늦게 온 소식
print('소포 수:', db.execute('SELECT count(*) FROM to_ship').fetchone()[0])
```

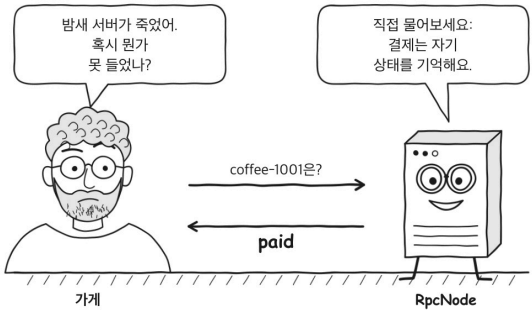
```
1 아직 결제가 아님
2 발송
3 이미 처리했거나 우리 것이 아님
4 아직 결제가 아님
소포 수: 1
```

소포는 하나예요. 모든 것이 명령 하나에 달려 있습니다. "`paid` 로 바꾸되, 주문이 아직 대기 중이고 결제와 금액이 바로 그것일 때만 바꿔라." 표시와 발송 작업은 함께 기록돼요. 둘 다 되거나, 아무것도 안 되거나. 이것을 데이터베이스 트랜잭션이라고 해요(말은 블록체인의 것과 같지만, 다른 것입니다). 똑같은 소식 여덟 개를 한꺼번에 넣어도 확인해 봤는데, 발송은 하나였어요.

26장의 대강의 코드에서는 전송 번호가 메모리에 있었고, 일을 하기 전에 기억해 버렸어요. 돈을 다룰 때는 그러면 안 됩니다. 프로그램이 중간에 죽으면 전부 뒤엉켜요. 먼저 데이터베이스에 적고, 그다음 200을 응답하세요.

소식이 아예 안 왔다

웹훅은 일곱 번 시도한 뒤 버려져요. 약 일곱 시간입니다. 소켓은 놓친 것을 아예 다시 보내지 않아요. 그러니 소식의 일부는 사라질 수 있습니다. 처방은 대조예요. 내 쪽에서 아직 열려 있는 주문을 직접 물어보는 거죠.



`GET /v1/payments/orders/{orderId}` 요청은 결제를 돌려줘요. 거기서는 이름이 조금 달라요. `event` 대신 `status` 를 보고, 네트워크는 `network_slug` , 토큰 컨트랙트는 `token_contract` 입니다. 응답으로 똑같은 모양의 '소식'을 만들어 같은 자물쇠에 통과시키세요.

서버가 멈췄다 돌아온 뒤, 소켓을 다시 연결한 뒤, 그리고 정해진 주기로 대조하세요. 열린 주문 10건을 분당 한 번이면, 허용된 100건 중 10건입니다.

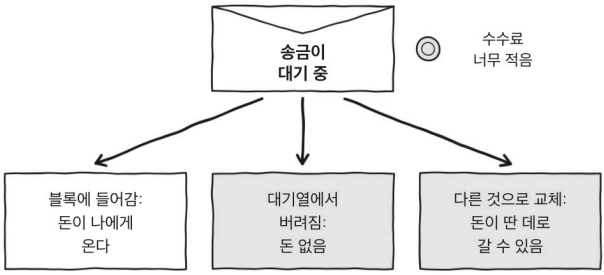
송금이 멈췄다

손님이 해시를 보냈어요. "한 시간 전에 보냈어요." 결제는 말이 없고요. 노드에게 그런 트랜잭션을 아는지 물어봅시다. 여기의 해시는 지어낸 거예요.

```
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` 은 "이 노드는 그런 것을 모른다"는 뜻이에요. 이유는 여러 가지입니다. 오타, 다른 네트워크, 아직 도착하지 않은 송금, 또는 이미 버려진 송금. 트랜잭션은 찾았는데 `blockNumber` 가 `null` 이면, 대기열에서 기다리는 중이에요.



송금이 블록에 없으면 돈도 아직 없어요

네트워크	오래 기다리는 송금에게 생기는 일
Ethereum과 비슷한 곳	보낸 사람의 뒤 송금들을 모두 붙잡음(15장). 같은 `nonce`에 더 큰 수수료의 송금으로 교체할 수 있음. geth 노드 프로그램은 기본적으로 최소 10% 더 높기를 요구
Bitcoin	보낸 사람이 더 큰 수수료의 송금으로 교체할 수 있음. Bitcoin Core 프로그램은 기본적으로 2주 묵은 것을 대기열에서 버림
TRON	트랜잭션에 유효 기간이 있고 기본값은 1분. 블록에 못 들어가면 이후에도 못 들어감
Solana	보통의 트랜잭션은 참조하는 블록이 신선한 동안만 유효: 150블록, 겨우 수십 초

가게는 대개 남의 송금을 빠르게 만들 수 없어요. 보낸 사람의 키가 필요하니까요. 블록을 기다리고, 발송하지 마세요. 그사이 결제가 만료됐는데 송금이 결국 처리되면, 돈은 내 주소에 들어오지만 결제는 `expired`로 남습니다.

내가 보낸 것, 예를 들어 환불이 멈췄나요? 서명한 같은 송금은 다시 보낼 수 있어요. 해시가 바뀌지 않습니다. 하지만 새 송금은, 첫 번째를 교체하는 것이 아니라면 두 번째 돈이에요. Ethereum에서 주소에 대기열이 있는지는 이렇게 봅니다.

```
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"}
```

첫 번째 숫자는 그 주소의 송금 중 이미 블록에 든 것의 수이고, 두 번째는 대기 중인 것까지 합친 수예요. 같으니, 이 노드에는 대기열이 보이지 않아요. 15장 송금의 보낸 사람 주소입니다.

그리고 해시 감시(27장)는 조심하세요. `cancelled`라는 대답은 서비스가 트랜잭션을 찾지 못했다는 뜻이지, 앞으로 절대 나타나지 않는다는 뜻이 아니에요.

돈은 왔는데, 이상하게 왔다

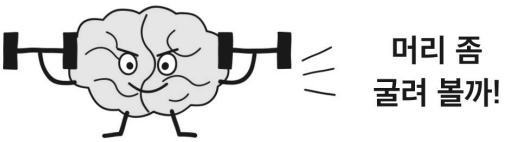
무슨 일	결제는 어떻게 되나
모자라게 옴	결제 완료 안 됨, 기한에 만료. 돈은 이미 나에게 있음
두 번에 나눠 옴	조각은 합쳐지지 않음: 위와 같음
더 많이 옴	`paid`. 실제 금액은 `received_amount_raw`
`paid` 뒤에 또 보냄	결제는 더 기다리지 않음: 주소 이벤트로나 알 수 있음
기한 뒤에 옴	`expired`
다른 토큰이나 다른 네트워크	인정 안 됨(16장)
트랜잭션 실패	돈이 없으니 발송 금지: 결제는 `failed`가 되거나 그냥 만료됨

경우마다 어떻게 할지 미리 정하고, 결제 페이지에 손님이 보도록 적어 두세요.

함정: "온 곳으로 돌려주지 뭐"

손님이 더 많이 냈고, 아르티옴은 남는 돈을 보낸 주소로 돌려주려 해요. 그런데 그곳은 거래소의 공용 주소일 수 있어요. 그러면 돈은 손님이 아니라 거래소가 받습니다. 손님에게 환불받을 주소와 네트워크를 물어보세요. 사이트에서가 아니라 지갑에서 직접 손으로 보내세요.

생각해 보기



1. 결제 만들기의 응답이 오지 않았어요. 다시 보내도 될까요? 어떤 `order_id`로요?
2. 똑같은 `paid`가 웹훅과 소켓으로 왔어요. 예제 프로그램은 소포를 몇 개 보낼까요?
3. 30 USDT 대신 15씩 두 번 보냈어요. 30짜리 결제는 어떻게 될까요?

정답: 1) 네, 같은 번호와 같은 데이터로요. 2) 하나. 3) 결제 완료가 안 돼요. 송금은 합쳐지지 않아요.

핵심 정리

- 다시 보내도 안전한 것만 다시 보내세요.
- 발송은 데이터베이스의 자물쇠가 지키고, 잃어버린 소식은 대조가 찾아 줘요.
- 송금이 블록에 없는 동안은 돈이 없는 거예요. 이상한 결제에 대한 규칙은 미리 필요해요.

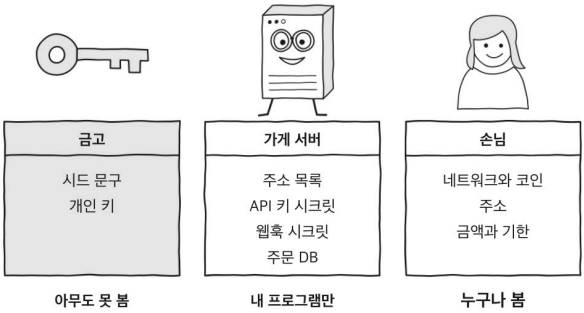
다음: 30장. 보안과 출시 전 체크리스트.

제30장. 보안과 출시 전 체크리스트

"내일 문 여는데, 뭘 빠뜨렸지?"

테스트넷에서는 모든 게 잘 돌아가요. 아르티옴은 벌써 손님들에게 "암호화폐 받습니다!"라고 알리고 있어요. 그런데 잠이 안 옵니다. 돈이 진짜가 되면 무엇이 잘못될 수 있을까요?

알아보기: 무엇이 어디에 있나



API 키의 시크릿으로는 여러분의 코인을 쓸 수 없어요. RpcNode에는 개인 키가 없으니까요. 하지만 말썽은 일으킬 수 있습니다. 키에 '허용 IP'를 지정하세요. 그러면 훔친 시크릿은 다른 컴퓨터에서 통하지 않아요.

2단계 인증은 비밀번호 외에 앱의 코드가 하나 더 필요한 거예요. 대시보드에서는 '설정'에 있습니다. 메일에도 똑같이 켜세요. 다른 모든 것의 접근 권한을 메일로 되찾는 경우가 많으니까요.

이 중 무언가가 새어 나가면 어떻게 될까요.

새어 나간 것	남이 할 수 있는 일	대처
시드 문구, 개인 키	돈을 영영 가져감	남은 돈을 급히 새 지갑으로 옮기기
API 키의 시크릿	결제, 감시, 엔드포인트, 웹훅을 마음대로 다루고 웹훅 시크릿을 읽음	키를 삭제하고 새로 발급, 웹훅 시크릿 교체, 설정 점검
웹훅 시크릿	`paid` 소식을 위조	'새 시크릿 생성'
엔드포인트 주소	내 크레딧을 씀	새로 만들고 옛것은 삭제(22장)
대시보드 로그인	내 코인을 빼고는 이 표의 거의 모든 것	비밀번호 변경, 2단계 인증 켜기

시크릿은 코드에 살지 않는다

코드 속의 시크릿은 공유 저장소로, 외주 업체에 보내는 압축 파일로, 화면 캡처로 실려 나가요. 시크릿은 서버 설정(환경 변수)에 두고, 그 파일은 외부인이 못 보게 막으세요.

직접 해보기: 내 것부터 뒤져 보기

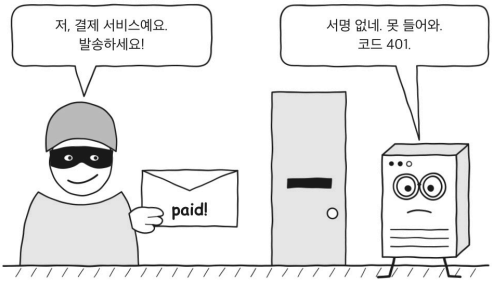
API 키의 시크릿은 `sk_live_`로 시작해요. 실수가 하나 든 연습용 폴더를 만들고 찾아봅시다.

```
mkdir demo
echo "const SECRET = 'sk_live_primer'" > 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_primer'
```

검색은 첫 번째 파일을 찾아냈고, 두 번째는 건드리지 않았어요. `demo`를 여러분의 프로젝트 폴더로 바꿔 보세요. `xprv`는 확장 개인 키의 시작이에요. 무언가 나왔다면 그 줄을 지우는 것만으로는 부족합니다. 변경 이력에 남아 있을 수 있어요. 시크릿이 새었다고 보고 새로 발급하세요. 그리고 시드 문구는 이렇게 잡을 수 없어요. 평범한 단어들이니까요. 아예 컴퓨터에 두지 마세요.

들어오는 것을 믿지 마세요



내 문을 점검해 봅시다. 26장의 수신기 코드를 `receiver.mjs`에 저장하고 실행하세요.

```
RPCNODE_WEBHOOK_SECRET=demo-secret node receiver.mjs
```

다른 터미널 창에서 남인 척 세 가지 방법으로 두드려 보세요.

```
URL=http://localhost:3000
BODY='{"type":"payment_order","event":"paid"}'
# 1) 서명 없이
curl -s -o /dev/null -w '%{http_code}\n' $URL -d "$BODY"
# 2) 아무렇게나 지어낸 서명
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) 서명은 진짜지만 한 시간 묵은 것
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"
```

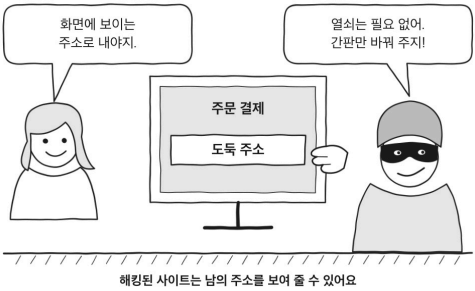
```
401
401
401
```

세 번 모두 "못 들어와"예요. 어딘가에서 200이 왔다면 문이 열린 겁니다. `- 3600`을 `- 0`으로 바꾸면 세 번째 두드림은 200을 받아요. 서명이 신선하고 맞으니까요.

비싼 주문에는 두 번째 자물쇠를 다세요. `paid`를 받으면 결제를 직접 물어보고(`GET /v1/payments/orders/{orderId}`) `payment_id`, 주소, 토큰, 금액, 상태를 대조하세요. 그러면 웹훅 시크릿을 훔쳐도 커피를 공짜로 가져갈 수 없어요.

또 웹훅은 `https`로 받고, 들어오는 요청의 크기를 제한하세요. 26장의 코드는 이것을 하지 않습니다. 덧붙여 '시크릿 및 IP' 항목의 주소만 들여보낼 수도 있어요. 이것은 서명에 더하는 것이지, 서명을 대신하는 것이 아닙니다.

바꿔치기한 간판



해킹된 사이트는 남의 주소를 보여 줄 수 있어요

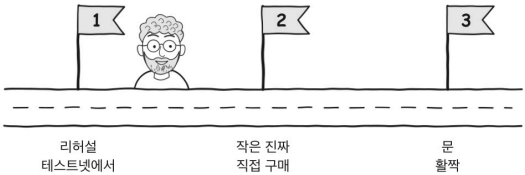
도둑이 꼭 키를 손에 넣어야 하는 것은 아니에요. 사이트를 해킹해 손님에게 자기 주소를 보여 주면 됩니다. 이런 것이 도움이 돼요.

- 사이트 관리자 화면에서 주소 목록을 몰래 바꿀 수 없게 하기.
- 가끔 다른 컴퓨터에서, 나눠 준 주소를 내 지갑과 대조하기.

- 때때로 직접 손님이 되어 사 보고, 어디로 내라고 하는지 보기.
- 서버의 프로그램을 업데이트하고, 관리자 화면에는 2단계 인증으로 들어가기.

도둑은 여러분에게 직접 말을 걸기도 해요. '고객센터', "멈춘 결제를 돌려주세요", 비슷한 주소에서 온 아주 작은 송금(6장). 진짜 고객센터는 시드 문구를 묻지 않아요.

리허설



먼저 전부 테스트넷에서 해요. 테스트넷의 결제와 이벤트는 TRON Nile에서 동작합니다(26장). Free에는 샌드박스가 있어요. 기본적으로 동시 결제 3건과 주소 3개이고, 거기의 위험 검사는 진짜가 아니에요. 샌드박스에서도 주소는 저절로 비지 않습니다.

연습할 상황	나와야 할 결과
정확히 결제	소포 하나
모자라게 결제	발송 없음, 결제 만료
같은 소식을 나에게 두 번 보내기	소포 하나
내 서버를 끄고, 결제하고, 다시 켜기	결제를 찾아냄: 웹훅 재전송이나 대조로
서명 없이 두드리기	401

그다음, 장사할 바로 그 네트워크에서 자기 자신에게 작은 진짜 구매를 한 번 하세요. 그 김에 지갑이 돈을 알아보는지, 그 돈을 다시 보낼 수 있는지도 확인하세요. 수수료 낼 코인이 충분한가요.

법과 세금

많은 나라에서 암호화폐를 받는 것은 제한되거나 과세돼요. 출시 전에 여러분 나라의 법을 확인하세요. 이것은 법률 자문이 아닙니다. 그 김에 회계사에게 어떤 결제 기록이 필요한지 묻고, 첫날부터 보관하세요.

함정: "한 번 확인했으니 잊자"

가게는 돌아가는데, 주변은 모두 변해요. 요금제 기간이 끝나면 계정은 Free로 넘어가고, 거기에는 메인넷 결제가 없습니다(25장). 크레딧은 이벤트에도 쓰여요. 주소 목록은 바닥을 향해 갑니다. 알림을 맞춰 두세요.

출시 전 체크리스트

키와 접근

- □ 시드 문구는 적어 두었고, 어떤 서버나 컴퓨터에도 없다.
- □ 사이트에는 주소만 있고, 그중 몇 개는 지갑과 대조했다.
- □ 코드에 시크릿이 없고, 프로젝트 검색 결과가 비어 있다.
- □ API 키에 허용 IP를 지정했다.
- □ 대시보드, 메일, 사이트 관리자 화면에 2단계 인증을 켰다.

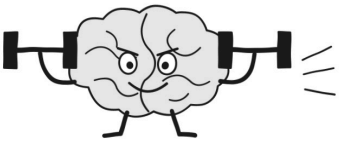
돈 받기

- □ 주문 하나에 주소 하나, 중복이 없다.
- □ 청구서에 네트워크, 코인, 금액, 주소, 기한, “한 번에 송금”이 있다.
- □ 토큰 컨트랙트는 `GET /v1/network-tokens`에서 가져와 발행사 사이트와 대조했다.
- □ 컨펌 수를 내 금액에 맞게 골랐다.
- □ 웹훅 서명을 검사하고, 데이터베이스의 자물쇠가 발송을 지킨다.
- □ 열린 주문의 대조가 정해진 주기로 돈다.

주변 일

- □ 부족 결제, 초과 결제, 지각, 환불에 대한 규칙이 있다.
- □ 슬롯, 크레딧, 요금제 기간, 남은 주소를 지켜본다.
- □ 돈을 모을 때 수수료를 무엇으로 낼지 알고 있다.
- □ 주문 데이터베이스를 백업하고, 한 번은 백업에서 복구해 봤다.
- □ 리허설을 마쳤고, 직접 진짜 구매를 해 봤다.
- □ 우리 나라의 법과 세금을 확인했다.

생각해 보기



머리 좀 굴려 볼까!

1. API 키의 시크릿이 새었어요. 남이 내 지갑의 코인을 빼 갈 수 있을까요? 그래도 당장 해야 할 일은?
2. 내 수신기가 서명 없는 메시지에 200을 응답했어요. 무엇이 위험할까요?
3. TRON Nile에서 모든 검사를 통과했어요. 새 검사 없이 Ethereum 결제를 열어도 될까요?

정답: 1) 아니요, 거기에는 지갑 키가 없어요. 하지만 API 키를 삭제하고 새로 발급하고, 웹훅 시크릿을 바꾸고, 설정을 점검해야 해요. 2) 누구든 돈 없이 주문을 '결제'할 수 있어요. 3) 안 돼요. 네트워크, 주소, 수수료, 컨펌이 다 달라요.

핵심 정리

- 돈의 키는 사이트에서 멀리, 시크릿은 코드에서 멀리 두세요.
- 들어오는 메시지도, 화면 캡처도, 메일도 말만 믿지 마세요.
- 먼저 리허설, 그다음 작은 진짜 구매, 그다음 개업.

파트 6 끝. 아르티옴은 가게를 조립했고, 고장 났을 때 무엇을 할지, 어떻게 지킬지 알게 됐어요.

다음: 31장. DeFi: 중개자 없는 금융.

제31장. DeFi: 중개자 없는 금융

“코인이 그냥 누워만 있어야 하나요?”

아르티옴에게 매출의 일부가 USDC로 쌓였어요. 아는 사람이 권합니다. “DeFi에 넣어 봐, 돈이 알아서 돈을 벌어.” 창구도 직원도 없는 그곳은 대체 뭘까요?

> 이것은 금융 조언이 아니라 설명입니다. 프로토콜 이름은 예시일 뿐이에요. 여러분의 나라에서는 이런 서비스가 제한되거나 과세될 수 있으니, 법을 확인하세요.

알아보기: 은행과 프로그램

DeFi, ‘탈중앙화 금융’은 스마트 컨트랙트(9장)가 수행하는 교환, 대출, 그 밖의 돈 서비스예요. 공통 규칙을 가진 컨트랙트 묶음을 프로토콜이라고 합니다.

	은행	DeFi
돈은 어디에	은행에	주소나 컨트랙트에
누가 처리하나	직원과 그 시스템	네트워크의 프로그램
언제 일하나	정해진 시간에	24시간
규칙은 어디에	계약서에	코드에
실수했다면	은행에 문의 가능	보통 취소해 줄 사람이 없음
접근 권한을 잃었다면	신분증이 도움이 됨	키 없이는 되찾지 못함

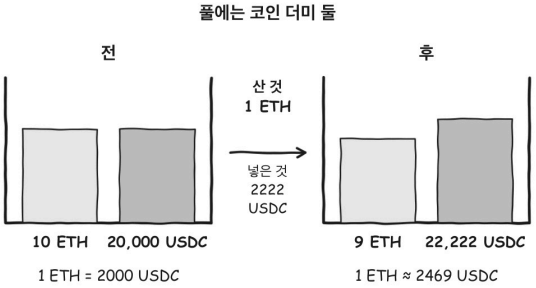
“중개자가 없다”가 “믿을 필요가 없다”는 뜻은 아니에요. 코드를, 코드를 바꿀 수 있는 사람을, 가격 공급자를, 코인 발행사를 믿는 것입니다.

거래소 없는 교환: 유동성 풀

보통의 거래소에서는 사려는 사람에게 팔 사람을 찾아 줘요. DeFi에서는 흔히 유동성 풀을 만듭니다. 두 가지 코인이 함께 들어 있는 컨트랙트예요. 풀에 하나를 넣고 다른 하나를 가져갑니다. 상대방을 기다릴 필요가 없어요.

얼마를 받을지는 공식이 정해요. 단순한 풀의 공식은 x × y = k이고, x와 y는 두 코인의 잔량입니다. 교환으로 이 곱이 줄어들면 안 돼요.

여기서 가격은 잔량의 비율이에요. 풀에 10 ETH와 20,000 USDC가 있다고 해 봅시다. 그러면 1 ETH는 약 2000 USDC이고, k = 200,000입니다. 아르티옴이 1 ETH를 가져가려 해요(수수료는 일단 빼고요). 9 ETH가 남으니, USDC는 200,000 ÷ 9 ≈ 22,222가 돼야 합니다.



그는 2000이 아니라 2222 USDC를 내게 돼요. 자기가 산 것 때문에 가격이 오른 거예요. 이것을 가격 영향(price impact)이라고 합니다. 백 배 큰 풀(1000 ETH와 2,000,000 USDC)이었다면 같은 1 ETH는 약 2002 USDC였을 거예요.

풀의 ETH가 다른 곳보다 비싸지면, 누군가 ETH를 가져와 USDC를 빼 가요. 가격이 가까워질 때까지요. 이것이 차익 거래(아비트라지)입니다.

풀에 코인을 넣는 사람은 유동성 공급자이고, 수수료의 일부를 받아요. 하지만 가격이 벌어지면 풀은 비싸진 코인을 내주고 다른 코인을 모읍니다. 수수료를 빼고 보면, 가격이 두 배 올랐을 때 예치한 것의 가치는 코인을 그냥 들고 있었을 때보다 약 5.7% 적어요. 이것을 비영구적 손실이라고 합니다. 이 순간에 예치를 빼면, 아주 영구적인 손실이 돼요.

직접 해보기: 진짜 풀에게 묻기

Ethereum의 Uniswap v2 프로토콜에서 USDC와 WETH 풀을 골라 볼게요. WETH는 토큰 형태의 ETH로, 일대일입니다.

풀 주소는 아무 검색에서나 가져오면 안 돼요. 풀 '팩토리' 주소는 Uniswap 문서에서, USDC 주소는 발행사 Circle에서, WETH 주소는 ethereum.org에서 가져왔습니다. 그런 다음 팩토리에게 `getPair`를 물었어요. "이 짝의 풀이 어디 있어?" 대답은 `0xB4e16d0168e52d35CaCD2c6185b44281Ec28C9Dc`입니다.

이제 `getReserves` 질문이에요. "잔량이 얼마야?" 9장의 `eth_call`이라서, 읽기만 합니다.

```
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"]}'
```

2026년 11월 27일로 넘어가는 밤에 노드는 이렇게 대답했어요.

```
{"jsonrpc":"2.0","id":1,"result":"0x000000000000000000000000000000000000000000000000079ecc8266bc000000000000000000000000000000000000000000000000010b762ae141ddef4f5000000000000000000000000000000000000000000000000000000006b08b4af"}
```

`0x` 뒤에 64자리 숫자 세 개가 이어져요.

숫자(앞의 0은 생략)	뜻
`79ecc8266bc`	USDC 잔량, 소수점 아래 6자리: 8,378,617 USDC
`10b762ae141ddef4f50`	WETH 잔량, 18자리: 4933.80 WETH
`6b08b4af`	잔량이 마지막으로 바뀐 시각, 1970년부터의 초

어느 코인이 먼저인지는 `token0`과 `token1` 질문이 알려 줘요. 이 풀의 첫 번째는 USDC입니다. 이제 나누기만 하면 돼요.

```
python3 -c 'u = 0x79ecc8266bc / 10**6; w = 0x10b762ae141ddef4f50 / 10**18; print(u, w, u / w)'
```

`8378617.325244 4933.79553339896 1698.2092728662094`가 출력됩니다.

이 풀에서 1 ETH는 약 1698 USDC였어요. 여러분의 숫자는 다를 거예요.

큰 물량이면 얼마를 줄까요? 여기의 교환 수수료는 0.3%예요. 그중 0.25%는 공급자에게, 0.05%는 프로토콜에게 갑니다. 같은 잔량으로 계산하면 이렇습니다.

내는 것	받는 USDC	1 ETH당	가격 1698.21보다 나쁜 정도
1 ETH	1692.77	1692.77	0.3%
100 ETH	165,957.86	1659.58	2.3%
1000 ETH	1,408,492.58	1408.49	17%

계산은 Uniswap의 보조 컨트랙트(router)에 `getAmountsOut`을 물은 대답과 마지막 자리까지 일치했어요.

중요: 잔량을 나누는 방법은 v2 방식의 풀에만 맞아요. Uniswap v3와 v4에서는 예치금이 가격 구간별로 나뉘어 있어서, 가격을 다르게 계산합니다.

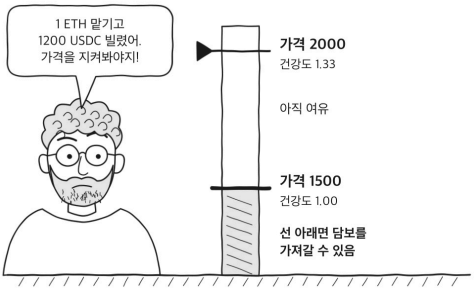
담보 대출과 청산

대출 프로토콜(예: Aave)에서는 담보를 넣고, 그 가치보다 적게 빌려요. 왜 그럴까요? 예를 들어 달러가 필요한데 ETH를 팔기는 싫은 경우예요.

대출에는 '건강도'(health factor)가 있어요.

> 건강도 = 담보 가치 × 청산 기준 ÷ 빚

지어낸 예예요. 담보는 개당 2000 USDC인 ETH 1개, 기준은 80%, 빚은 1200 USDC. 건강도는 2000 × 0.8 ÷ 1200 ≈ 1.33입니다. 가격이 1500으로 떨어지면 딱 1이 돼요.



건강도가 1보다 낮으면 누구든 빚의 일부, 때로는 전부를 대신 갚고, 그 대가로 웃돈을 얹어 담보를 가져갈 수 있어요. 이것이 청산입니다. "돈을 더 넣으세요"라는 전화는 오지 않아요. 한밤중이라도 프로그램이 다 해 버립니다. 빚은 이자로 저절로 늘기도 해요.

스테이블코인: '1달러'를 무엇이 떠받치나

9장에서 봤듯, 스테이블코인은 1달러 언저리 값을 유지하려 할 뿐이에요. 떠받치는 방식은 여러 가지입니다. 발행사의 준비금: 달러와 그 밖의 자산(USDT, USDC). 컨트랙트 안의 넉넉한 암호화폐 담보(DAI, USDS. 담보에 다른 스테이블코인이 들기도 해요). 담보 없이 코인 개수를 바꾸는 알고리즘.

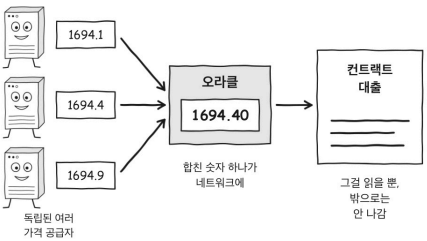
어느 방식도 보장은 아니에요. 풀 안의 이런 코인이 싸지면, 공급자에게는 바로 그 코인이 쌓입니다.

스테이킹 짧게 보기

8장의 검증자에게는 32 ETH가 필요해요. 더 적은 금액으로는 공동 스테이킹 풀을 통해 참여합니다(교환 풀이 아니에요). 대신 보관증 토큰을 주는 경우가 많아요. 위험은 검증자에 대한 벌금, 컨트랙트의 오류, 운영자에 대한 믿음입니다. 수익이 얼마일지는 아무도 약속하지 않아요.

오라클: 컨트랙트는 가격을 어떻게 알까

대출 컨트랙트에는 ETH 가격이 필요해요. 하지만 스스로 인터넷에 나가지는 않습니다. 모든 노드가 같은 결과를 얻어야 하니까요(15장). 가격은 오라클이 네트워크에 적어 넣어요.



예는 Chainlink 오라클이에요. 'ETH / USD' 피드 주소는 Chainlink의 안내 자료에서 가져왔습니다. `latestRoundData` 질문은 최신 기록을 달라고 해요.

```
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":"0x0000000000000000000000000000000000000000000000070000000000008e4f00000000000000000000000000000000000000000000000000000000277369380000000000000000000000000000000000000000000000000000000000006b08aefa000000000000000000000000000000000000000000000000000006b08af0f000000000000000000000000000000000000000000000000000007000000000008e4f"}
```

다섯 숫자 중 두 번째인 `2773693800`이 소수점 아래 8자리의 가격이에요. 1694.40달러입니다. 풀은 1698.21 USDC를 보여 줬으니, 차이는 약 0.2%예요. 네 번째 숫자는 기록 시각이고, 우리가 묻기 25분 전이었어요. 이 피드는 가격이 적힌 값에서 0.5% 벗어날 때, 그리고 정해진 주기(약 한 시간에 한 번)로 갱신됩니다.

왜 풀에서 바로 가격을 가져오지 않을까요? 큰 거래로 잠깐 가격을 밀어 버릴 수 있기 때문이에요. 하지만 오라클도 틀리거나 늦을 수 있고, 청산은 오라클의 숫자대로 진행됩니다.

DeFi 앱은 네트워크를 어떻게 읽을까

교환이나 대출 사이트는 진열장이에요. 데이터는 노드에게서 가져옵니다.

- `eth_call`: 지금 어떤지. 풀 잔량, 잔액, 오라클 가격, 대출 건강도.
- 이벤트(15장): 무엇이 바뀌었는지. v2 풀은 바뀔 때마다 새 잔량을 담은 `Sync` 이벤트를 발행해요. `eth_getLogs`로 찾거나 WebSocket으로 듣습니다(11장).
- 전송: 교환은 장부를 바꿔요. 지갑이 서명하고, 노드가 전달하며(7장), 가스가 듭니다.

우리 풀의 최신 블록 `Sync` 이벤트를 묻는 질문이에요.

```
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":[]}
```

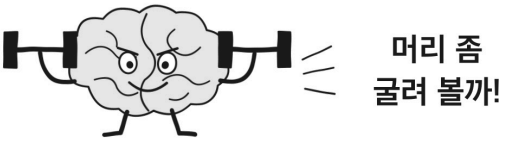
빈 목록도 대답이에요. 이 블록에서는 풀이 바뀌지 않았습니다. `getReserves`의 세 번째 숫자가 이를 확인해 줘요. 마지막 변화는 우리가 묻기 약 10분 전이었어요. 지난 변화, 예를 들어 가격 그래프용 데이터는 이미 아카이브나 자체 데이터베이스의 몫입니다(10장).

함정: 무엇이 잘못될 수 있나

위험	벌어지는 일	위험을 줄이는 법
코드의 오류	돈을 도둑맞거나 갇힘	코드 감사. 하지만 보장은 아님
가짜 토큰	이름은 익숙한데 주소는 남의 것	주소 대조(9장)
무제한 허용(`approve`)	나중에 토큰을 가져감	조금만 허용하고, 남는 허용은 0으로
슬리피지	기대보다 적게 받음	최소 금액 지정
청산	웃돈까지 얹혀 담보가 사라짐	넉넉한 여유와 지켜보기
쌍둥이 사이트	사이트 복제본이 서명을 요구	주소 확인, 시드 문구는 입력 금지
'확실한 수익'	약속하고 사라짐	참여하지 않기

슬리피지는 기대한 가격과 실제 가격의 차이예요. 교환이 대기열에서 기다리는 동안 남의 거래가, 때로는 일부러, 가격을 밀어요. 최소 금액이 보호해 줍니다. 그보다 나쁘면 교환이 이루어지지 않아요. 가스는 쓰이지만요.

생각해 보기



1. 풀에 100 ETH와 200,000 USDC가 있어요. 누군가 20 ETH를 가져갔어요. USDC를 얼마나 넣었을까요(수수료 제외)?
2. 담보는 개당 2000 USDC인 ETH 2개, 기준 80%, 빚 2400 USDC. 건강도는 얼마일까요? ETH 가격이 얼마면 1이 될까요?
3. 어떤 사이트가 위험 없이 하루 3%를 약속하며 USDT를 제한 없이 쓰게 허용해 달라고 해요. 어떻게 할까요?

정답: 1) 80 ETH가 남으니 USDC는 250,000이 돼요. 50,000을 넣었고, ETH당 2500이에요. 2) 약 1.33. 가격이 1500일 때. 3) 아무것도 서명하지 말고 사이트를 닫아요.

핵심 정리

- DeFi에서는 컨트랙트가 서비스를 수행하고, 위험과 책임은 상당 부분 여러분에게 와요.
- v2 방식 풀에서 가격은 두 잔량에서 나오고, 큰 거래는 가격을 움직여요.
- 앱은 여러분과 똑같은 요청, `eth_call`과 이벤트로 네트워크를 읽어요.

다음: 맺음말. 손님의 편지에서 내 가게까지.

2026년 11월 27일 확인: 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.

맺음말. 손님의 편지에서 내 가게까지

모든 것은 메일의 한 줄에서 시작됐어요. “암호화폐로 결제해도 될까요?” 그때 아르티옴은 노드가 뭔지도 모르고 지갑을 손으로 확인했죠. 여러분이 그와 함께 어떤 길을 걸어왔는지 돌아보세요.

이제 아르티옴이 할 수 있는 일

파트	배운 것
1. 제공업체가 필요한 이유	자체 서버 비용을 계산하고, 노드를 직접 돌지 제공업체에게 맡길지 고르기
2. 처음부터 배우는 블록체인	블록, 키, 서명, 수수료, 컨펌, 토큰 이해하기
3. 노드와 RPC	노드에게 질문하고, 노드의 종류를 구별하고, 뒤처지지 않았는지 확인하기
4. 네트워크는 어떻게 동작할까	네트워크의 차이를 알아보고, 손님에게 맞는 것 고르기
5. RpcNode	엔드포인트를 받고, 호출 가격을 계산하고, 묻지 않고도 결제를 알기
6. 실습	결제 받기를 조립하고, 장애를 견디고, 가게를 지키기
7. 다음 단계: DeFi	네트워크에서 풀을 직접 읽고, 위험을 솔직하게 따지기

기억해 둘 일곱 가지 원칙

1. 주소는 누구에게나 보여 주고, 개인 키와 시드 문구는 아무에게도 보여 주지 않는다.
2. 네트워크, 토큰 컨트랙트 주소, 받는 사람, 금액을 한꺼번에 확인한다.
3. 돈이 도착한 것은 화면 캡처가 왔을 때가 아니라, 트랜잭션이 성공하고 컨펌이 쌓였을 때다.
4. 200 코드는 아직 성공이 아니다. 응답 안을 읽는다.
5. 소식이 몇 번 오든, 결제 하나에 발송은 한 번.
6. 엔드포인트와 시크릿은 비밀번호처럼 간수한다.
7. 말만 믿지 않는다. 네트워크에 묻고 대답을 확인한다.

내일은 무엇부터

1. 3장의 공개 노드 요청을 다시 해 보세요.
2. 나만의 엔드포인트와 테스트넷을 마련하세요(22장).
3. 장난감 돈으로 자기 자신에게 결제를 연습해 보세요(28장과 30장).
4. 30장의 체크리스트를 통과하세요.
5. 여러분 나라의 법과 세금을 확인하세요.

낡은 내용이 보인다면

이 책은 2026년 11월 27일의 네트워크와 서비스 모습을 담았어요. 네트워크는 업데이트되고, 가격과 요금제는 바뀝니다. 노드의 대답이 책과 다르다면, 노드와 최신 문서를 믿으세요. RpcNode의 요금제와 기능은 rpcnode.dev 사이트에서 확인하세요.

안녕히

매출의 일부를 DeFi에 넣을지, 아르티옴은 아직 고민 중이에요. 대신 어떤 질문을 해야 하는지는 알게 됐죠. 어쩌면 이것이 이 이야기 전체에서 그가 얻은 가장 큰 것일 거예요.

이 책의 숫자 일부는 낡겠지만, 확인하는 습관은 낡지 않아요. 여러분과 여러분의 일에 행운을 빕니다!



부록 A. 용어 사전

괄호 안은 그 용어를 더 자세히 설명한 장이에요.

가스 — Ethereum과 친척 네트워크에서 수수료를 매기는 작업량의 단위 (15).

개인 키 — 돈을 다루는 데 쓰는 것. 아무에게도 보여 주지 않음 (6).

검증자(밸리데이터) — 담보를 맡기고 블록 생성에 참여하는 참가자 (8).

노드 — 네트워크 프로그램이 도는 컴퓨터. 보관하고, 검증하고, 대답함 (1).

동기화 — 노드가 네트워크를 따라잡는 과정 (12).

멤풀 — 블록을 기다리는 송금의 대기열 (7).

멱등성 — 몇 번을 반복해도 결과가 한 번 한 것과 같은 성질 (29).

부하 분산(밸런싱) — 여러 노드에 요청을 나눠 주는 것 (23).

브리지 — 가치를 한 네트워크에서 다른 네트워크로 옮기는 방법 (16).

블록 — '장부의 한 페이지': 트랜잭션, 번호, 시각, 이전 블록의 도장 (5).

블록체인 — 공통 규칙에 따라 함께 관리하는 공용 기록 목록 (4).

서명 — 키의 주인이 송금을 허락했다는 증명 (6).

수수료 — 블록의 자리에 대해 네트워크에 내는 값 (7).

스마트 컨트랙트 — 블록체인에 적힌 프로그램 (9).

스테이블코인 — 1달러 언저리를 유지하려는 토큰 (9).

스테이킹 — 코인을 담보로 맡기고 네트워크를 지키는 데 참여하는 것 (8).

시드 문구 — 키를 복구할 수 있는 12개 또는 24개의 단어 (6).

아카이브 노드 — 과거 어느 블록 시점의 상태든 기억하는 노드 (10).

엔드포인트 — 프로그램이 노드에게 질문하는 주소 (3).

오라클 — 가격 같은 바깥 정보를 네트워크에 적어 넣는 쪽 (31).

요청 한도(RPS) — 1초에 몇 번 물을 수 있는지 (3, 25).

웹훅 — 일이 생기면 서비스가 내 주소로 직접 보내는 요청 (26).

유동성 풀 — 두 코인이 들어 있고 공식에 따라 교환해 주는 컨트랙트 (31).

이벤트(log) — 무슨 일이 있었는지 컨트랙트가 알리는 것 (15).

이중 지불 — 동전 하나로 두 번 값을 치르려는 시도 (4).

인덱서 — 블록체인 기록을 '카드함'에 정리해 두는 프로그램 (10).

제공업체 — 노드를 운영하고 접근할 수 있게 해 주는 서비스 (1, 3).

주소 — 돈을 받는 '우편함 번호' (6).

지갑 — 키를 보관하는 프로그램이나 기기 (6).

채굴(마이닝) — 블록을 붙일 권리를 두고 대입으로 문제를 푸는 것 (8).

청산 — 담보가 더는 대출을 받쳐 주지 못할 때 담보를 처분하는 것 (31).

체인 재구성(리오그) — 최근 블록 일부가 다른 가지로 바뀌는 것 (8).

컨펌 — 내 트랜잭션 위에 쌓인 블록. 트랜잭션이 담긴 블록도 셈에 넣음 (5).

크레딧 — 서비스가 요청 사용량을 세는 단위 (3, 25).

테스트넷 — 값어치 없는 코인으로 연습하는 곳 (22).

토큰 — 스마트 컨트랙트가 관리하는 기록 (9).

파이널리티(최종 확정) — 그 뒤로는 블록이 바뀌지 않는 시점 (8).

프루닝 노드 — 모두 검증한 뒤 오래된 블록을 지운 노드 (10).

합의(콘센서스) — 참가자들이 하나의 체인을 고르는 규칙 (8).

해시, '도장' — 어떤 데이터든 짧게 나타내는 지문 (5).

AML — 돈이 나쁜 이력과 연결돼 있지 않은지 보는 검사 (27).

API — 한 프로그램이 다른 프로그램에게 말을 거는 창구 (11).

Chain ID — EVM 네트워크의 번호 (16).

DeFi — 스마트 컨트랙트가 수행하는 돈 서비스 (31).

EVM — 컨트랙트를 실행하는 Ethereum 가상 머신 (15).

gRPC — 프로그램이 엄격한 스키마의 압축 메시지로 대화하는 방식 (11).

L2, '2층' — 송금은 자기 안에서 처리하고 그 기록을 Ethereum으로 보내는 네트워크 (16).

Nonce — 계정의 트랜잭션에 붙는 순번 (15).

REST — 질문마다 주소 안의 경로가 따로 있는 방식 (11).

RPC, JSON-RPC — 다른 컴퓨터의 프로그램에게 무언가를 해 달라고 부탁하는 방식 (11).

UTXO — Bitcoin과 친척 네트워크의 쓰지 않은 '돈 조각' (14).

WebSocket — 소식을 받기 위해 닫지 않는 연결 (11).

Wei — ETH의 가장 작은 단위 (1).

부록 B. 한 페이지로 보는 책 속 네트워크

6장, 8장, 14–20장의 내용이며, 2026년 11월 27일 기준이에요. 네트워크 업데이트에 따라 바뀝니다.

Bitcoin과 친척들 (14장)

네트워크	코인	블록 간격(대략)	주소의 시작
Bitcoin	BTC	10분	`1`, `3`, `bc1`
Litecoin	LTC	2.5분	`L`, `M`, `ltc1`
Dogecoin	DOGE	1분	`D`
Bitcoin Cash	BCH	10분	`bitcoincash:q…`
Dash	DASH	2.5분	`X`
Zcash	ZEC	75초	`t1`, `zs`, `u1`

Ethereum과 EVM 네트워크 (15장과 16장)

모두 주소가 `0x`로 시작해요.

네트워크	Chain ID	수수료 코인	블록 간격(대략)
Ethereum	1	ETH	12초
Base	8453	ETH	0.2초
Arbitrum One	42161	ETH	0.3초
BNB Smart Chain	56	BNB	0.45초
Mantle	5000	MNT	2초
Monad	143	MON	0.3초

그 밖의 네트워크 (17–20장)

네트워크	코인	주소의 시작	최종 확정
TRON	TRX	`T`	약 1분
Solana	SOL	특별한 시작 글자 없음	1초 미만
TON	Gram	`EQ`, `UQ`	약 1초
Sui	SUI	`0x`	1초 미만
Aptos	APT	`0x`	1초 미만
XRP Ledger	XRP	`r`	3–5초
Stellar	XLM	`G`	약 5초
Cardano	ADA	`addr1`	서서히
Polkadot Hub	DOT	`1`	표시로 확인
Cosmos Hub	ATOM	`cosmos1`	빠름

부록 C. 요청 치트 시트

모든 예제는 RpcNode의 공개 노드에서 동작해요. 가입도 키도 필요 없지만, 요청 빈도에 제한이 있습니다. 여러분의 프로젝트에는 주소를 개인 엔드포인트로 바꾸세요.

JSON-RPC: 명령 하나로 모두

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

표의 메서드와 매개변수를 넣으세요. 다른 네트워크는 주소의 `ethereum`을 바꾸면 됩니다.

Ethereum과 EVM 네트워크

알고 싶은 것	메서드	매개변수
마지막 블록 번호	`eth_blockNumber`	없음
네트워크 번호	`eth_chainId`	없음
코인 잔액	`eth_getBalance`	`"0x주소","latest"`
트랜잭션 영수증	`eth_getTransactionReceipt`	`"0x해시"`
가스 가격	`eth_gasPrice`	없음
컨트랙트에게 묻기	`eth_call`	`{"to":"0x…","data":"0x…"},"latest"`

Bitcoin과 Solana

주소	메서드	알 수 있는 것
`public-bitcoin-mainnet`	`getblockcount`	체인 높이
`public-bitcoin-mainnet`	`getblockchaininfo`	노드 상태
`public-solana-mainnet`	`getSlot`	슬롯 번호
`public-solana-mainnet`	`getBalance`	램포트 단위 잔액, 매개변수는 주소

TRON과 TON: 질문이 주소 안에

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

```
curl https://public-ton-mainnet.rpcnode.dev/api/v2/getMasterchainInfo
```

오류 코드

코드	무슨 일
`-32700`	요청을 해석할 수 없음
`-32601`	메서드를 찾을 수 없거나 쓸 수 없음
`-32602`	잘못된 매개변수
`-32005` , HTTP 429	너무 자주 요청함
`-32006` , HTTP 402	크레딧 부족

전체 문서와 최신 요금제

https://rpcnode.dev

Telegram에서 만나요

@rpcnode_dev

지은이: rpcnode.dev

끝.

쉬운 말로

안개 걷힌 블록체인

이 책에서 무엇을 배울까요?

손님이 물어요. “암호화폐로 결제해도 되나요?” 뭐라고 답하고, 돈이 왔는지 어떻게 알고, 중간에 잃지 않을까요?

담긴 내용: 블록, 키, 수수료와 토큰, 노드와 요청, Bitcoin, Ethereum, TRON, Solana, TON 등의 네트워크. 이어서 실습: RpcNode 엔드포인트, 결제, 웹훅, 가게 지키기. 마지막에는 DeFi에 관한 솔직한 이야기.

“내 사이트는
장바구니도 계산하고
메일도 보내요.
그런데 블록체인
속의 일은
어떻게 알죠?”

— **아르티옴,**
이 책의 주인공



이 책은 뭐가 특별할까?

미리 알아야 할 것은 없어요. 장마다 주인공의 질문, 쉬운 답, 그림과 과제. 요청은 직접 실행해 보고 진짜 응답을 볼 수 있어요.

rpcnode.dev
docs.rpcnode.dev
Telegram: @rpcnode_dev
지은이: rpcnode.dev