---

title: Nodes and clients description: An overview of Ethereum nodes and client software, plus how to set up a node and why you should do it. lang: en sidebarDepth: 2

Ethereum is a distributed network of computers (known as nodes) running software that can verify blocks and transaction data. The software must be run on your computer to turn it into an Ethereum node. There are two separate pieces of software (known as 'clients') required to form a node.

Prerequisites {#prerequisites}

You should understand the concept of a peer-to-peer network and thebasics of the EVM before diving deeper and running your own instance of an Ethereum client. Take a look at our introduction to Ethereum.

If you're new to the topic of nodes, we recommend first checking out our user-friendly introduction onrunning an Ethereum node.

What are nodes and clients? {#what-are-nodes-and-clients}

A "node" is any instance of Ethereum client software that is connected to other computers also running Ethereum software, forming a network. A client is an implementation of Ethereum that verifies data against the protocol rules and keeps the network secure. A node has to run two clients: a consensus client and an execution client.

- The execution client (also known as the Execution Engine, EL client or formerly the Eth1 client) listens to new transactions broadcasted in the network, executes them in EVM, and holds the latest state and database of all current Ethereum data.
- The consensus client (also known as the Beacon Node, CL client or formerly the Eth2 client) implements the proof-of-stake consensus algorithm, which enables the network to achieve agreement based on validated data from the execution client. There is also a third piece of software, known as a 'validator' that can be added to the consensus client, allowing a node to participate in securing the network.

These clients work together to keep track of the head of the Ethereum chain and allow users to interact with the Ethereum network. The modular design with multiple pieces of software working together is called encapsulated complexity. This approach made it easier to execute The Merge seamlessly, makes client software easier to maintain and develop, and enables the reuse of individual clients, for example, in the layer 2 ecosystem.

 Simplified diagram of a coupled execution and consensus client.

Client diversity {#client-diversity}

Both execution clients and consensus clients exist in a variety of programming languages developed by different teams.

Multiple client implementations can make the network stronger by reducing its dependency on a single codebase. The ideal goal is to achieve diversity without any client dominating the network, thereby eliminating a potential single point of failure. The variety of languages also invites a broader developer community and allows them to create integrations in their preferred language.

Learn more about client diversity.

What these implementations have in common is they all follow a single specification. Specifications dictate how the Ethereum network and blockchain functions. Every technical detail is defined and specifications can be found as:

- Originally, the Ethereum Yellow Paper
- Execution specs
- Consensus specs
- EIPs implemented in various network upgrades

Tracking nodes in the network {#network-overview}

Multiple trackers offer a real-time overview of nodes in the Ethereum network. Note that due to the nature of decentralized networks, these crawlers can only provide a limited view of the network and might report different results.

- Map of nodes by Etherscan
- Ethernodes by Bitfly
- Ethereum Node Crawler
- Nodewatch by Chainsafe, crawling consensus nodes

Node types {#node-types}

If you want to run your own node, you should understand that there are different types of node that consume data differently. In fact, clients can run three different types of nodes: light, full and archive. There are also options of different sync strategies which enable faster synchronization time. Synchronization refers to how quickly it can get the most up-to-date information on Ethereum's state.

Full node {#full-node}

Full nodes do a block-by-block validation of the blockchain, including downloading and verifying the block body and state data for each block. There are different classes of full node - some start from the genesis block and verify every single block in the entire history of the blockchain. Others start their verification at a more recent block that they trust to be valid (e.g. Geth's 'snap sync'). Regardless of where the verification starts, full nodes only keep a local copy of relatively recent data (typically the most recent 128 blocks), allowing older data to be deleted to save disk space. Older data can be regenerated when it is needed.

- Stores full blockchain data (although this is periodically pruned so a full node does not store all state data back to genesis)
- Participates in block validation, verifies all blocks and states.
- All states can be either retrieved from local storage or regenerated from 'snapshots' by a full node.
- Serves the network and provides data on request.

Archive node {#archive-node}

Archive nodes are full nodes that verify every block from genesis and never delete any of the downloaded data.

- Stores everything kept in the full node and builds an archive of historical states. It is needed if you want to query something like an account balance at block #4,000,000, or simply and reliably test your own transactions set without mining them using tracing.
- This data represents units of terabytes, which makes archive nodes less attractive for average users but can be handy for services like block explorers, wallet vendors, and chain analytics.

Syncing clients in any mode other than archive will result in pruned blockchain data. This means, there is no archive of all historical states but the full node is able to build them on demand.

Learn more about Archive nodes.

Light node {#light-node}

Instead of downloading every block, light nodes only download block headers. These headers contain summary information about the contents of the blocks. Any other information the light node requires gets requested from a full node. The light node can then independently verify the data they receive against the state roots in the block headers. Light nodes enable users to participate in the Ethereum network without the powerful hardware or high bandwidth required to run full nodes. Eventually, light nodes might run on mobile phones or embedded devices. The light nodes do not participate in consensus (i.e. they cannot be miners/validators), but they can access the Ethereum blockchain with the same functionality and security guarantees as a full node.

Light clients are an area of active development for Ethereum and we expect to see new light clients for the consensus layer and execution layer soon. There are also potential routes to providing light client data over the gossip network. This is advantageous because the gossip network could support a network of light nodes without requiring full nodes to serve requests.

Ethereum does not support a large population of light nodes yet, but light node support is an area expected to develop rapidly in the near future. In particular, clients like Nimbus, Helios, and LodeStar are currently heavily focused on light nodes.

Why should I run an Ethereum node? {#why-should-i-run-an-ethereum-node}

Running a node allows you to directly, trustlessly and privately use Ethereum while supporting the network by keeping it more robust and decentralized.

Benefits to you {#benefits-to-you}

Running your own node enables you to use Ethereum in a private, self-sufficient and trustless manner. You don't need to trust the network because you can verify the data yourself with your client. "Don't trust, verify" is a popular blockchain mantra.

- Your node verifies all the transactions and blocks against consensus rules by itself. This means you don’t have to rely on any other nodes in the network or fully trust them.
- You can use an Ethereum wallet with your own node. You can use dapps more securely and privately because you won't have to leak your addresses and balances to intermediaries. Everything can be checked with your own client. MetaMask, Frame, and many other wallets offer RPC-importing, allowing them to use your node.
- You can run and self-host other services which depend on data from Ethereum. For example, this might be a Beacon Chain validator, software like layer 2, infrastructure, block explorers, payment processors, etc.
- You can provide your own custom RPC endpoints. You could even offer these endpoints publicly to the community to help them avoid big centralized providers.
- You can connect to your node using **Inter-process Communications (IPC)** or rewrite the node to load your program as a plugin. This grants low latency, which helps a lot, e.g. when processing a lot of data using web3 libraries or when you need to replace your transactions as fast as possible (i.e. frontrunning).
- You can directly stake ETH to secure the network and earn rewards. See solo staking to get started.

Network benefits {#network-benefits}

A diverse set of nodes is important for Ethereum’s health, security and operational resiliency.

- Full nodes enforce the consensus rules so they can’t be tricked into accepting blocks that don't follow them. This provides extra security in the network because if all the nodes were light nodes, which don't do full verification, validators could attack the network.
- In case of an attack which overcomes the crypto-economic defenses of proof-of-stake, a social recovery can be performed by full nodes choosing to follow the honest chain.
- More nodes in the network result in a more diverse and robust network, the ultimate goal of decentralization, which enables a censorship-resistant and reliable system.
- Full nodes provide access to blockchain data for lightweight clients that depend on it. Light nodes don't store the whole blockchain, instead they verify data via the state roots in block headers. They can request more information from full nodes if they need it.

If you run a full node, the whole Ethereum network benefits from it, even if you don't run a validator.

Running your own node {#running-your-own-node}

Interested in running your own Ethereum client?

For a beginner-friendly introduction visit our run a node page to learn more.

If you're more of a technical user, dive into more details and options on how to spin up your own node.

Alternatives {#alternatives}

Setting up your own node can cost you time and resources but you don’t always need to run your own instance. In this case, you can use a third party API provider. For an overview of using these services, check out nodes as a service.

If somebody runs an Ethereum node with a public API in your community, you can point your wallets to a community node via Custom RPC and gain more privacy than with some random trusted third party.

On the other hand, if you run a client, you can share it with your friends who might need it.

Execution clients {#execution-clients}

The Ethereum community maintains multiple open-source execution clients (previously known as 'Eth1 clients', or just 'Ethereum clients'), developed by different teams using different programming languages. This makes the network stronger and more diverse. The ideal goal is to achieve diversity without any client dominating to reduce any single points of failure.

This table summarizes the different clients. All of them passclient tests and are actively maintained to stay updated with network upgrades.

| Client | Language | Operating systems | Networks | Sync strategies | State pruning | | ---------- | ---------- | ---------- | ---------- | ---------- | ---------- | | Geth | Go | Linux, Windows, macOS | Mainnet, Sepolia, Goerli | Snap, Full | Archive, Pruned | | Nethermind | C#, .NET | Linux, Windows, macOS | Mainnet, Sepolia, Goerli, and more | Snap (without serving), Fast, Full | Archive, Pruned | | Besu | Java | Linux, Windows, macOS | Mainnet, Sepolia, Goerli, and more | Snap, Fast, Full | Archive, Pruned | | Erigon | Go | Linux, Windows, macOS | Mainnet, Sepolia, Goerli, and more | Full | Archive, Pruned | | Reth | Rust | Linux, Windows, macOS | Mainnet, Sepolia, Goerli, and more | Full | Archive, Pruned |

For more on supported networks, read up onEthereum networks.

Each client has unique use cases and advantages, so you should choose one based on your own preferences. Diversity allows implementations to be focused on different features and user audiences. You may want to choose a client based on features, support, programming language, or licences.

Besu {#besu}

Hyperledger Besu is an enterprise-grade Ethereum client for public and permissioned networks. It runs all of the Ethereum Mainnet features, from tracing to GraphQL, has extensive monitoring and is supported by ConsenSys, both in open community channels and through commercial SLAs for enterprises. It is written in Java and is Apache 2.0 licensed.

Besu's extensive documentation will guide you through all details on its features and setups.

Erigon {#erigon}

Erigon, formerly known as Turbo-Geth, started as a fork of Go Ethereum oriented toward speed and disk-space efficiency. Erigon is a completely re-architected implementation of Ethereum, currently written in Go but with implementations in other languages under development. Erigon's goal is to provide a faster, more modular, and more optimized implementation of Ethereum. It can perform a full archive node sync using around 2TB of disk space, in under 3 days.

Go Ethereum {#geth}

Go Ethereum (Geth for short) is one of the original implementations of the Ethereum protocol. Currently, it is the most widespread client with the biggest user base and variety of tooling for users and developers. It is written in Go, fully open source and licensed under the GNU LGPL v3.

Learn more about Geth in itsdocumentation.

Nethermind {#nethermind}

Nethermind is an Ethereum implementation created with the C# .NET tech stack, licensed with LGPL-3.0, running on all major platforms including ARM. It offers great performance with:

- an optimized virtual machine
- state access
- networking and rich features like Prometheus/Grafana dashboards, seq enterprise logging support, JSON RPC tracing,

and analytics plugins.

Nethermind also has detailed documentation, strong dev support, an online community and 24/7 support available for premium users.

Consensus clients {#consensus-clients}

There are multiple consensus clients (previously known as 'Eth2' clients) to support theconsensus upgrades. They are responsible for all consensus-related logic including the fork-choice algorithm, processing attestations and managing proof-of-stake rewards and penalties.

| Client | Language | Operating systems | Networks | | ---------- | ---------- | ---------- | ---------- | | Lighthouse | Rust | Linux, Windows, macOS | Beacon Chain, Goerli, Pyrmont, Sepolia, Ropsten, and more | | Lodestar | TypeScript | Linux, Windows, macOS | Beacon Chain, Goerli, Sepolia, Ropsten, and more | | Nimbus | Nim | Linux, Windows, macOS | Beacon Chain, Goerli, Sepolia, Ropsten, and more | | Prysm | Go | Linux, Windows, macOS | Beacon Chain, Gnosis, Goerli, Pyrmont, Sepolia, Ropsten, and more | | Teku | Java | Linux, Windows, macOS | Beacon Chain, Gnosis, Goerli, Sepolia, Ropsten, and more |

Lighthouse {#lighthouse}

Lighthouse is a consensus client implementation written in Rust under the Apache-2.0 license. It is maintained by Sigma Prime and has been stable and production-ready since Beacon Chain genesis. It is relied upon by various enterprises, staking pools and individuals. It aims to be secure, performant and interoperable in a wide range of environments, from desktop PCs to sophisticated automated deployments.

Documentation can be found in Lighthouse Book

Lodestar {#lodestar}

Lodestar is a production-ready consensus client implementation written in Typescript under the LGPL-3.0 license. It is maintained by ChainSafe Systems and is the newest of the consensus clients for solo-stakers, developers and researchers. Lodestar consists of a beacon node and validator client powered by JavaScript implementations of Ethereum protocols. Lodestar aims to improve Ethereum usability with light clients, expand accessibility to a larger group of developers and further contribute to ecosystem diversity.

More information can be found on ourLodestar website

Nimbus {#nimbus}

Nimbus is a consensus client implementation written in Nim under the Apache-2.0 license. It is a production-ready client in use by solo-stakers and staking pools. Nimbus is designed for resource efficiency, making it easy to run on resource-restricted devices and enterprise infrastructure with equal ease, without compromising stability or reward performance. A lighter resource footprint means the client has a greater margin of safety when the network is under stress.

Learn more inNimbus docs

Prysm {#prysm}

Prysm is a full-featured, open source consensus client written in Go under the GPL-3.0 license. It features an optional webapp UI and prioritizes user experience, documentation, and configurability for both stake-at-home and institutional users.

Visit Prysm docs to learn more.

Teku {#teku}

Teku is one of the original Beacon Chain genesis clients. Alongside the usual goals (security, robustness, stability, usability, performance), Teku specifically aims to comply fully with all the various consensus client standards.

Teku offers very flexible deployment options. The beacon node and validator client can be run together as a single process,

which is extremely convenient for solo stakers, or nodes can be run separately for sophisticated staking operations. In addition, Teku is fully interoperable with Web3Signer for signing key security and slashing protection.

Teku is written in Java and is Apache 2.0 licensed. It is developed by the Protocols team at ConsenSys that is also responsible for Besu and Web3Signer. Learn more in Teku docs.

Synchronization modes {#sync-modes}

To follow and verify current data in the network, the Ethereum client needs to sync with the latest network state. This is done by downloading data from peers, cryptographically verifying their integrity, and building a local blockchain database.

Synchronization modes represent different approaches to this process with various trade-offs. Clients also vary in their implementation of sync algorithms. Always refer to the official documentation of your chosen client for specifics on implementation.

Execution layer sync modes {#execution-layer-sync-modes}

Full archive sync {#full-sync}

Full sync downloads all blocks (including headers, transactions, and receipts) and generates the state of the blockchain incrementally by executing every block from genesis.

- Minimizes trust and offers the highest security by verifying every transaction.
- With an increasing number of transactions, it can take days to weeks to process all transactions.

Full snap sync {#snap-sync}

Snap sync verifies the chain block-by-block, just like a full archive sync; however, instead of starting at the genesis block, it starts at a more recent 'trusted' checkpoint that is known to be part of the true blockchain. The node saves periodic checkpoints while deleting data older than a certain age. Those snapshots are used to regenerate state data when it is needed, rather than having to store it all forever.

- Fastest sync strategy, currently default in Ethereum mainnet
- Saves a lot of disk usage and network bandwidth without sacrificing security

More on snap sync

Light sync {#light-sync}

Light client mode downloads all block headers, block data, and verifies some randomly. Only syncs tip of the chain from the trusted checkpoint.

- Gets only the latest state while relying on trust in developers and consensus mechanism.
- Client ready to use with current network state in a few minutes.

NB Light sync does not yet work with proof-of-stake Ethereum - new versions of light sync should ship soon!

More on light clients

Consensus layer sync modes {#consensus-layer-sync-modes}

Optimistic sync {#optimistic-sync}

Optimistic sync is a post-merge synchronization strategy designed to be opt-in and backwards compatible, allowing execution nodes to sync via established methods. The execution engine can *optimistically* import beacon blocks without fully verifying them, find the latest head, and then start syncing the chain with the above methods. Then, after the execution client has caught up, it will inform the consensus client of the validity of the transactions in the Beacon Chain.

More on optimistic sync

Checkpoint sync {#checkpoint-sync}

Checkpoint sync, also known as weak subjectivity sync, creates a superior user experience for syncing Beacon Node. It's based on assumptions of weak subjectivity which enables syncing Beacon Chain from a recent weak subjectivity checkpoint instead of genesis. Checkpoint sync makes the initial sync time significantly faster with similar trust assumptions as syncing from genesis.

In practice, this means your node connects to a remote service to download recent finalized states and continues verifying data from that point. Third party providing the data is trusted and should be picked carefully.

More on checkpoint sync

Further reading {#further-reading}

There is a lot of information about Ethereum clients on the internet. Here are few resources that might be helpful.

- Ethereum 101 - Part 2 - Understanding Nodes– *Wil Barnes, 13 February 2019*
- Running Ethereum Full Nodes: A Guide for the Barely Motivated– *Justin Leroux, 7 November 2019*

Related topics {#related-topics}

- Blocks
- Networks

Related tutorials {#related-tutorials}

- Turn your Raspberry Pi 4 into a validator node just by flashing the MicroSD card – Installation guide– *Flash your Raspberry Pi 4, plug in an ethernet cable, connect the SSD disk and power up the device to turn the Raspberry Pi 4 into a full Ethereum node running the execution layer (Mainnet) and / or the consensus layer (Beacon Chain / validator).*