Ethereum nodes

Ethereum is a distributed network of computers (known as nodes) running software that can verify blocks and transaction data. An ad hoc software must be run to turn the computer into an Ethereum node. 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. After ethereum moved from proof of work to proof of stake 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.

Local node vs provider

This paper is written processing data from a self hosted node. That is not mandatory as it is possible to rely on services created ad hoc. **Infura** and **Alchemy** are popular third-party services that provide easy access to Ethereum nodes via APIs, allowing developers to interact with the Ethereum blockchain without running their own node. These services abstract the complexities of node management and provide scalable infrastructure for dApps, smart contract deployments, and blockchain analytics.

Using such service has the advantage of providing a standardised environment out of the box but comes with some disadvantages. On a bulk analysis API calls can get expensive, there is not a full control on the data accessed, the providers can block access and there are centralisation risks which are exacerbated by relying fully on third party providers.

The main point here is the lack of control on data while a node implementation gives both the control and tools for accessing such data. Being this paper based on data analysis we prioritised the choice of a local node provider suitable to self hosting.

Running a node is not trivial at the current stage to the ethereum network. An archive node can require more than 10 TB disk space with certain implementations, a good bandwidth, fast disks and a computer with 32 Gb Ram and multicore.

Ethereum’s network is composed of a decentralized group of validation nodes that since the Merge follow PoS to secure the blockchain. The Merge shifted Ethereum from PoW to PoS, where validators propose and validate blocks instead of miners.

These validators use both execution layer and consensus layer clients (such as Geth, Erigon, Prysm, and Lighthouse) to keep the Ethereum network operational and secure. The capacity to choose different client implementations allows for network diversity and resilience.

Comparison of Full Nodes vs Archive Nodes in Various Ethereum Implementations

Ethereum nodes come in different types based on the amount of data they store and the roles they serve in the network. Running a self hosted node to perform data analysis requires to choose between a full node and an archive node.

A full node stores the complete current state of the Ethereum blockchain (i.e., account balances, contract storage, etc.) and recent historical data, but it prunes old state data to save space. Full nodes verify all transactions and blocks from the genesis block to the current state but don't keep every historical state like past balances or contract storage at every block.

An Archive Node stores everything that a full node does, but in addition, it keeps all historical states for every block in the blockchain. This means archive nodes can provide the exact state of the blockchain at any point in history, but they require significantly more disk space. Full nodes are sufficient for most operations, while archive nodes are only necessary if you need access to the entire historical state of the blockchain.

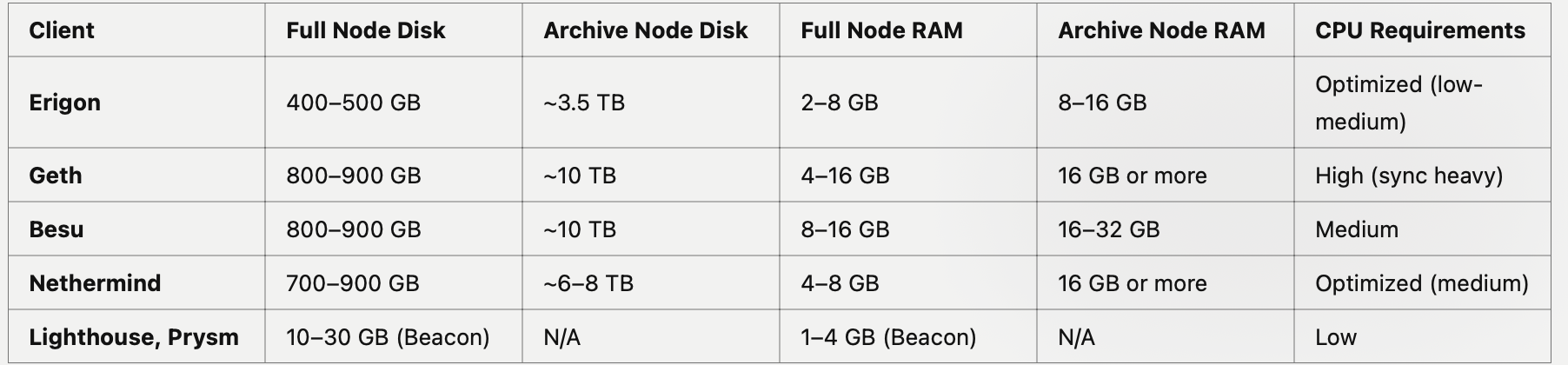
A full node stores enough information to partecipate to the ethereum network, also as validator, but data analysis would become cumbersome as the stored information is enough to rebuild and validate the current state but requires computation to do it. Therefore in this paper the archive node is the best choice. There are many implementations of full archive nodes and our requirements for data analysis are quite clear: save disk space, fast access to data, a client which allow to present the data in an human readable format for easy interpretation. We tested the most commonly used to decide the best fit.

Geth is the most widely used Ethereum client, Erigon is designed to be more resource-efficient than traditional Geth, especially in terms of disk usage and sync time.

Besu is an Ethereum client written in Java, commonly used in enterprise environments.

Nethermind is a high-performance Ethereum client written in C# with a focus on speed and configurability.

Besu and Geth may require more memory depending on the workload.



I tested Geth plus Lighthouse (the latter only for consensus) and interrupted Geth and Besu while the second SSD disk installed as volume was half filled (max capacity 10 Tb and strong performance degradation). I didn’t test Nevermind as, being written in C#,

Motivation for choosing Erigon

Erigon looks the best fit as allows to run an archive node with a standard 4tb ssd disk. Being IO demanding two SSD in raid-0 are a good choice but not mandatory. Another advantage of Erigon is having an embedded consensus node implementation. Since the merge, when ethereum became proof of stake, there is the need of an execution node and a consensus node but for data analysis the simplest is the consensus the best. This is the case of caplin, the embedded consensus node in erigon, as it allows to achieve an archive node full synch without any configuration or installation of a compatible consensus node.

Before starting the work in the main chain we tested Erigon on sepolia, a light test network, and verified that the data saved in a key value db, mdbx, were properly indexed and rendered by the embedded UI, otterscan. The last requirement for the analysis is being able to stop the synch at a certain block and run an npc server which serves the data without synching continuously. Erigon satisfy also this. This allows to synch just once the main network till the desired block and then stop this operation, extremely resource demanding in term of bandwidth even when close to the ethereum chain tip.

An additional section addresses otterscan compared to other ethereum UI like etherscan block scout and so on. It explains also the difficulty in interpreting the raw data without a third party software as, in our case, otterscan.