---

title: Ethereum Archive Node description: An overview of archive nodes lang: en sidebarDepth: 2

An archive node is an instance of an Ethereum client configured to build an archive of all historical states. It is a useful tool for certain use cases but might be more tricky to run than a full node.

Prerequisites {#prerequisites}

You should understand the concept of anEthereum node, its architecture, sync strategies, practices of running and using them.

What is an archive node

To grasp the importance of an archive node, let's clarify the concept of "state." Ethereum can be referred to as *transaction-based state machine*. It consists of accounts and applications executing transactions which are changing their state. The global data with information about each account and contract is stored in a trie database called state. This is handled by the execution layer (EL) client and includes:

- Account balances and nonces
- Contract code and storage
- Consensus-related data, e.g. Staking Deposit Contract

To interact with the network, verify and produce new blocks, Ethereum clients have to keep up with the most recent changes (the tip of the chain) and therefore the current state. An execution layer client configured as a full node verifies and follows the latest state of the network but only caches the past few states, e.g. the state associated with the last 128 blocks, so it can handle chain reorgs and provide fast access to recent data. The recent state is what all clients need to verify incoming transactions and use the network.

You can imagine the state as a momentary network snapshot at a given block and the archive as a history replay.

Historical states can be safely pruned because they are not necessary for the network to operate and it would be uselessly redundant for client to keep all out-of-date data. States that existed before some recent block (e.g. 128 blocks before the head) are effectively thrown away. Full nodes only keep historical blockchain data (blocks and transactions) and occasional historical snapshots they can use to regenerate older states on request. They do this by re-executing past transactions in the EVM, which can be computationally demanding when the desired state is far from the nearest snapshot.

However, this means that accessing a historical state on a full node consumes a lot of computation. The client might need to execute all past transactions and compute one historical state from genesis. Archive nodes solve this by storing not only the most recent states but every historical state created after each block. It basically makes a trade-off with bigger disk space requirement.

It's important to note that the network does not depend on archive nodes to keep and provide all historical data. As mentioned above, all historical interim states can be derived on a full node. Transactions are stored by any full node (currently less than 400G) and can be replayed to build the whole archive.

Use cases

Regular usage of Ethereum like sending transactions, deploying contracts, verifying consensus, etc. does not require access to historical states. Users never need an archive node for a standard interaction with the network.

The main benefit of state archive is a quick access to queries about historical states. For example, archive node would promptly return results like:

- *What was ETH balance of account 0x1337... at block 15537393?*
- *What is the balance of token 0x in contract 0x at block 1920000?*

As explained above, a full node would need to generate this data by EVM execution which uses the CPU and takes time.

Archive nodes access them on the disk and serve responses immediately. This is a useful feature for certain parts of infrastructure, for example:

- Service providers like block explorers
- Researchers
- Security analysts
- Dapp developers
- Auditing and compliance

There are various free services that also allow access to historical data. As it is more demanding to run an archive node, this access is mostly limited and works only for occasional access. If your project requires constant access to historical data, you should consider running one yourself.

Implementations and usage

Archive node in this context means data served by user facing execution layer clients as they handle the state database and provide JSON-RPC endpoints. Configuration options, sync time and database size may vary by client. For details, please refer to the documentation provided by your client.

Before starting your own archive node, learn about the differences between the clients and especially the varioushardware requirements. Most clients are not optimized for this feature and their archives require more than 12TB of space. In contrast, implementations like Erigon can store the same data in under 3TB which makes them the most effective way of running an archive node.

Recommended practices

Apart from general recommendations for running a node, an archive node may be more demanding on hardware and maintenance. Considering Erigons key features, the most practical approach is using the Erigon client implementation.

Hardware

Always make sure to verify hardware requirements for a given mode in a client's documentation. The biggest requirement for archive nodes is the disk space. Depending on client, it varies from 3TB to 12TB. Even if HDD might be considered a better solution for large amounts of data, syncing it and constantly updating the head of the chain will require SSD drives. SATA drives are good enough but it should be a reliable quality, at leastTLC. Disks can be fitted into a desktop computer or a server with enough slots. Such dedicated devices are ideal for running high uptime node. It's totally possible to run it on a laptop but the portability will come at an additional cost.

All of the data needs to fit in one volume, therefore disks have to be joined, e.g. withRAID0 or LVM. It might be also worth considering using ZFS as it supports "Copy-on-write" which ensures data is correctly written to the disk without any low level errors.

For more stability and security in preventing accidental database corruption, especially in a professional setup, consider using ECC memory if your system supports it. The size of RAM is generally advised to be the same as for a full node but more RAM can help speed up the synchronization.

During initial sync, clients in archive mode will execute every transaction since genesis. Execution speed is mostly limited by the CPU, so a faster CPU can help with the initial sync time. On an average consumer computer, the initial sync can take up to a month.

Further reading {#further-reading}

- Ethereum Full Node vs Archive Node - *QuickNode, September 2022*
- Building Your Own Ethereum Archive Node - *Thomas Jay Rush, August 2021*
- How to set up Erigon, Erigon's RPC and TrueBlocks (scrape and API) as services– *Magnus Hansson, updated September 2022*

Related topics {#related-topics}

- Nodes and clients
- Running a node