APPENDIX

Start erigon in sepolia

Start erigon in main network

Start otterscan

Synchronisation phases

options YOU FIND ALL HERE

https://erigon.gitbook.io/erigon/advanced-usage/options

HOW I START ERiGON AND OTTERSCAN

TEST network (sepolia)

http enabled by def and keep ports

embedded services started as well as caplin

./build/bin/erigon --internalcl --datadir="\_sepoliadata" --chain=sepolia --ws --http.api=eth,debug,net,trace,web3,erigon --prune=hrtc (prune takes only last 90k blocks)

PROD

./build/bin/erigon --internalcl --datadir="ethereum" --ws --http.api=eth,debug,net,trace,web3,erigon add also (,ots --http.corsdomain "\*" --http.vhosts="\*" --http.addr=0.0.0.0) to use otterscan

In TEST TO CALL THE DOWNLOADED DATA STOPPING SYNCHING

./build/bin/rpcdaemon --datadir="\_sepoliadata" --http.api=eth,erigon,net,web3,ots --http.corsdomain="\*" (I see no reason for the following: --http.addr=0.0.0.0 --http.port=8545)

docker run --rm -p 5100:80 --name otterscan -d otterscan/otterscan:v2.6.0

**now otterscan can be accessed at http://127.0.0.1:5100/ --- important to start rpcdaemon as showed above.**

--private.api.addr=x.x.x.x:9090 erigon listen on 9090 (127.0.0.1) for gRPC call. USED to start independently services like RPCDAEMON (also txpool, rpcdaemon, sentry, downloader)

--http.api=eth,erigon,web3,net,debug,trace,txpool,parity,admin offered over http-rpc (default eth,erigon,engine)

--http.addr value normally i don't set it! HTTP-RPC server listening interface (default: "localhost")

--http.port value HTTP-RPC server listening port (default: 8545).

So normally I call this as localhost:8545 with a normal RPC call via curl or postman.

USEFUL

The Engine API in Erigon, and more broadly in Ethereum clients, refers to a set of JSON-RPC methods designed to support the interaction between the consensus layer (formerly known as Eth2) and the execution layer (formerly known as Eth1). This API is crucial for the operation of Ethereum after the network's transition from Proof of Work (PoW) to Proof of Stake (PoS) following "The Merge."

--authrpc.addr value HTTP-RPC server listening interface for the Engine API (default: "localhost")

--authrpc.port value HTTP-RPC server listening port for the Engine API (default: 8551)

--authrpc.jwtsecret value Path to the token that ensures safe connection between CL and EL

For web sockets

enough to pass --ws without params

reason

--ws.port value WS-RPC server listening port (default: 8546)

--ws Enable the WS-RPC server (default: false)

--ws.compression Enable compression over WebSocket (default: false)

docker...

note that I start manually the docker daemon with systctl

1. sudo systemctl start docker

then I run the container starting from a downloaded image

2. docker run --rm -p 5100:80 --name otterscan -d otterscan/otterscan:v2.6.0

flage used

The -d flag runs the container in detached mode (in the background).

The -p 5100:80 option maps port 5100 on your local machine to port 80 inside the container.

The --rm flag automatically removes the container when it exits.

The --name otterscan assigns a name to the running container.

3. docker ps

verifies running processes

This is the JSON response for eth.syncing, which provides detailed information about the synchronization process of an Ethereum node. Below is a breakdown and commentary on each field:

Main Fields:

currentBlock: 0: The block number that the node has currently processed. Since this is 0, it indicates that the node has not yet processed any blocks.

highestBlock: 20399999: The block number that the node is trying to reach. This represents the latest block in the blockchain at the time of syncing.

startingBlock: 0: The block number from which the synchronization started. Here, it's 0, which likely means the node started syncing from the genesis block.

Healing Fields (Related to state healing):

healedBytecodeBytes: 0 and healedBytecodes: 0: These show the number of bytecodes (and their size in bytes) that have been healed (repaired or corrected) during the sync process. In this case, no bytecodes have been healed.

healedTrienodeBytes: 0 and healedTrienodes: 0: These indicate the number of trienodes (data structures used in the Ethereum state tree) and their size in bytes that have been healed. No trienodes have been healed yet.

healingBytecode: 0 and healingTrienodes: 0: The number of bytecodes and trienodes currently being healed (actively in progress). In this case, there is no healing happening.

Sync Stages (Specific sync stages the node is going through):

The stages array provides detailed progress on various synchronization tasks:

stage\_name: "Snapshots": Refers to syncing of snapshots, which is a faster way of syncing the state.

block\_number: "0x137477f": The block number at which the snapshot sync is currently happening (hexadecimal for block 20399999).

stage\_name: "Headers": The synchronization of block headers. It's at block 0x137477f, which is the current block number (block 20399999).

stage\_name: "BorHeimdall": This is specific to sidechains like Polygon (Matic), suggesting the node is involved in syncing with the Polygon network, particularly the Bor layer. No progress here (block\_number: "0x0").

Other stages (e.g., "BlockHashes", "Bodies", "Senders"): These stages represent the processing of different components of blocks:

"BlockHashes": Verifying block hashes.

"Bodies": Syncing block bodies (transactions, logs).

"Senders": Syncing sender information for transactions.

stage\_name: "Execution": The node is executing block transactions at block 0x66ff1f (hexadecimal for block 6733087). The node is lagging behind in execution compared to other stages.

Stages with block\_number: "0x0": These stages haven't started yet. This includes:

Translation: Related to converting between different formats.

HashState: Syncing the state tree's hash.

AccountHistoryIndex, StorageHistoryIndex, LogIndex: Indexing historical account and storage data, logs.

CallTraces, TxLookup: Tracing calls and transactions.

stage\_name: "Finish": The final stage of synchronization, which hasn't started (block\_number: "0x0").

Additional Sync Stats:

syncedAccountBytes: 0, syncedAccounts: 0: No account data has been synced yet.

syncedBytecodeBytes: 0, syncedBytecodes: 0: No bytecode data has been synced yet.

syncedStorage: 0, syncedStorageBytes: 0: No storage data has been synced.

txIndexFinishedBlocks: 0, txIndexRemainingBlocks: 0: No blocks have been indexed for transactions, and all blocks still need to be indexed.

net.peerCount: Number of connected peers.

eth.blockNumber: Current block number.

eth.getBlock('latest'): Retrieve the latest block.

eth.getTransaction('0xtransactionhash'): Get transaction details by hash.

eth.syncing: Check sync status. Showed above a big response from the node

eth.getBalance('0xaccountaddress'): Get balance of an address.

eth.gasPrice: Get current gas price.

eth.mining: Check if node is mining.

The synch process for a full node including indexing for Erigon takes time and resources. Here’s a detailed breakdown of each step in the Erigon sync process, focusing on disk space requirements. Note as the indexing process will make easier the lookups for tools like otterscan.

1. BlockDownload (1/12)

Description: In this step, the node downloads block headers, block bodies, and transaction data from other peers.

Disk Space: This step is one of the most disk-intensive. Full blocks contain transaction data, state roots, receipts, and logs.

Estimate: For a full node (not archival), this can take around 1-2 TB of disk space. The exact amount depends on whether you’re keeping full historical data or using a pruned mode. For an archival node, this require some terabytes more.

2. BlockExecution (2/12)

Description: Executes transactions within blocks to recreate the state of the blockchain (account balances, smart contract states, etc.).

Disk Space: This step generates new state data but doesn’t significantly increase disk usage compared to BlockDownload. The state changes are tracked but don’t drastically grow storage needs.

Estimate: Minimal increase in space, since it primarily reuses the downloaded block data and updates the state.

3. Senders (3/12)

Description: Identifies the sender of each transaction and indexes this information for later lookups.

Disk Space: The data required for sender indexing is relatively small because the index primarily consists of mappings from transaction hashes to senders.

Estimate: Small—around a few gigabytes, depending on how many transactions have been processed on the blockchain.

4. Execution (4/12)

Description: Similar to BlockExecution, this step processes and executes blocks but includes more advanced state updates like gas usage, storage interactions, and more. It ensures that the node’s state is consistent with the chain.

Disk Space: Although computationally intensive, this step doesn’t drastically increase storage. It’s mainly concerned with executing transactions that have already been downloaded and updating the in-memory state.

Estimate: Minimal additional space, since it works mostly with the existing block data.

5. HashState (5/12)

Description: This step involves processing changes to account states, including balance updates and smart contract changes.

Disk Space: This can be moderately disk-intensive because it involves the account state changes. However, the size of account data is small relative to the entire blockchain.

Estimate: Moderate space usage, typically in the range of a few gigabytes to tens of gigabytes, depending on how many accounts are updated.

Steps Remaining:

6. TrieAccount (6/12)

Description: Builds the Merkle Patricia Trie for account state data, allowing efficient verification of account balances and states.

Disk Space: Compact and efficient in storage but can still grow due to the number of accounts on Ethereum.

Estimate: Minimal space increase, usually a few gigabytes.

7. TrieStorage (7/12)

Description: Builds the Merkle Patricia Trie for contract storage. This is more involved when smart contracts have large storage needs.

Disk Space: More storage-intensive than TrieAccount due to contract data, but still not as large as raw transaction data.

Estimate: Moderate space usage, depending on the amount of smart contract data (a few to several gigabytes).

8. ByteCodes (8/12)

Description: Processes and stores the bytecodes of smart contracts.

Disk Space: Smart contract bytecodes are generally small, even though there are many of them.

Estimate: Small space usage, typically a few gigabytes.

9. CallTraces (9/12)

Description: Processes call traces of transactions, which includes internal contract calls and transactions between contracts.

Disk Space: Call traces can get moderately large, especially for contracts with complex interactions, but it's still less than block data.

Estimate: Moderate space usage, potentially in the range of tens of gigabytes.

10. Logs (10/12)

Description: Processes and indexes event logs emitted by smart contracts. These logs are key for monitoring smart contract events.

Disk Space: Event logs can add up if many contracts are emitting frequent events, but they are generally lightweight.

Estimate: Small to moderate—likely a few gigabytes depending on the activity of smart contracts.

11. Receipts (11/12)

Description: Processes transaction receipts, which contain information about transaction outcomes (e.g., success/failure, gas usage, etc.).

Disk Space: Receipts are small data structures, and while there are many of them, the space needed is minimal.

Estimate: Small—a few gigabytes at most.

12. TxLookup (12/12)

Description: Creates an index to allow fast transaction lookups by hash or block number.

Disk Space: Indexes are typically efficient in terms of space usage but still require additional storage.

Estimate: Small—a few gigabytes, depending on the number of transactions.

Summary of Disk Space Usage by Step:

Step Disk Space Usage Estimate

1/12 BlockDownload High ~1-2 TB (Full node), more for archival nodes

2/12 BlockExecution Low Minimal increase in disk space

3/12 Senders Low A few gigabytes

4/12 Execution Low Minimal increase

5/12 HashState Moderate A few to several gigabytes

6/12 TrieAccount Low A few gigabytes

7/12 TrieStorage Moderate A few to several gigabytes

8/12 ByteCodes Low A few gigabytes

9/12 CallTraces Moderate Tens of gigabytes (depending on activity)

10/12 Logs Low to Moderate A few gigabytes

11/12 Receipts Low A few gigabytes

12/12 TxLookup Low A few gigabytes

Overall Estimate for a Full Node:

Total Disk Space for a full node: Around 2-3 TB for all steps combined, depending on whether pruning is enabled and the specific state of the Ethereum blockchain.

For an archival node, the space requirements would be significantly larger (typically close to 5 TB).

Most of the heavy disk space usage happens during the BlockDownload (1/12) phase. The remaining steps (2/12 to 12/12) are more about processing, indexing, and compact storage, and while they take time, they don’t consume nearly as much additional disk space as the first step.