NEAR Lake Framework

GitHub repo https://github.com/near/near-lake-framework/

Description

NEAR Lake Framework is an ecosystem of library companions toNEAR Lake . They allow you to build your own indexer that subscribes to the stream of blocks from theNEAR Lake data source and create your own logic to process the NEAR Protocol data in the programming languages of your choice (at the moment, there are implementations inPython ,JavaScript , andRust).

NEAR Lake Framework announcement We have announced the release of NEAR Lake Framework on NEAR Governance Forum.

Please, read the postthere .

How does it compare to NEAR Indexer Framework

?

Feature Indexer Framework Lake Framework Allows to follow the blocks and transactions in the NEAR Protocol Yes Yes

(but only mainnet and testnet networks) Decentralized Yes No (Pagoda Inc dumps the blocks to AWS S3) Reaction time (end-to-end) minimum 3.8s (estimated average 5-7s) minimum 3.9s (estimated average 6-8s) Reaction time (framework overhead only) 0.1s 0.2-2.2s Estimated cost of infrastructure 500+/mo 20/mo Ease of maintenance Advanced (need to follow every nearcore upgrade, and sync state) Easy

(deploy once and forget) How long will it take to start? days (on mainnet/testnet) seconds Ease of local development Advanced (localnet is a good option, but testing on testnet/mainnet is too heavy) Easy

(seetutorials) Programming languages that a custom indexer can be implemented with Rust only Any

(currently, helper packages are released inPython ,JavaScript , andRust)

Limitations

Lake Framework relies on the data being dumped to AWS S3 fromNEAR Lake Indexer which is based onNEAR Indexer Framework . Thus, Lake Framework is centralized around AWS S3 storage and also around maintainers of NEAR Lake Indexer nodes (Pagoda Inc). This is the tradeoff you might still want to take given all the other benefits mentioned above.

Indexers based on the Lake Framework inheritthe latency characteristics of Indexer Framework and extra latency of dumping to and reading from AWS S3, which is estimated to add at least 50ms delay while writing to S3, and 50ms on polling and reading from S3 (to make the polling cost-efficient, we default to polling only every 2s, so in the worst case you may observe an additional 2-second latency). Thus, Lake Framework adds 0.1-2.1s latency on top of Indexer Framework. Yet, again, most of the latency is there due to the finalization delay and both Indexer Framework and Lake Framework add quite a minimum overhead.

What is the cost of running a custom indexer based on NEAR Lake?

Indexers based on NEAR Lake consume 100-500MB of RAM depending on the size of the preloading queue, it does not require any storage, and it can potentially run even on Raspberry PI.

Getting the blockchain data from S3 will cost around 18.15 per month as NEAR Lake configured S3 buckets in such a way that the reader is paying the costs.

AWS S3 cost breakdown

Assuming NEAR Protocol produces exactly 1 block per second (which is really not, the average block production time is 1.3s). A full day consists of 86400 seconds, that's the max number of blocks that can be produced.

According theAmazon S3 prices list requests are charged for 0.005 per 1000 requests andget is charged for 0.0004 per 1000 requests.

Calculations (assuming we are following the tip of the network all the time):

86400 blocks per day * 5 requests for each block / 1000 requests * 0.0004 per 1k requests = 0.173 * 30 days = 5.19 Note: 5 requests for each block means we have 4 shards (1 file for common block data and 4 separate files for each shard)

And a number oflist requests we need to perform for 30 days:

86400 blocks per day / 1000 requests * 0.005 per 1k list requests = 0.432 * 30 days = 12.96

5.19 + 12.96 = 18.15 Note, the price depends on the number of shards.

Examples & Tutorials

- near-examples/near-lake-raw-printer
- : simple example of a data printer built on top of NEAR Lake Framework
- near-examples/near-lake-accounts-watcher
- : source code for a video tutorial on how to use the NEAR Lake Framework
- near-examples/indexer-tx-watcher-example-lake
- indexer example that watches for transaction for specified accounts/contracts build on top of NEAR Lake Framework

Tutorials SeeTutorials page Edit this page Last updatedonOct 19, 2022 byDamián Parrino Was this page helpful? Yes No

Previous Indexer Framework Next Runtime