EN components

On This Page * [API *][Fetcher *][State Keeper / VM *][Reorg Detector *][Consistency Checker *][Health check server *]

#

EN components

This section contains an overview of the EN's main components.

#

API

The EN can serve both the HTTP and the WS Web3 API, as well as PubSub. Whenever possible, it provides data based on the local state, with a few exceptions:

- Submitting transactions: Since it is a read replica, submitted transactions are proxied to the main node, and the response is returned from the main node.
- Querying transactions: The EN is not aware of the main node's mempool, and it does not sync rejected transactions. Therefore, if a local lookup for a transaction or its receipt fails, the EN will attempt the same query on the main node.

Apart from these cases, the API does not depend on the main node. Even if the main node is temporarily unavailable, the EN can continue to serve the state it has locally.

#

Fetcher

The Fetcher component is responsible for maintaining synchronization between the EN and the main node. Its primary task is to fetch new blocks in order to update the local chain state. However, its responsibilities extend beyond that. For instance, the Fetcher is also responsible for keeping track of L1 batch statuses. This involves monitoring whether locally applied batches have been committed, proven, or executed on L1.

It is worth noting that in addition to fetching thestate , the EN also retrieves the L1 gas price from the main node for the purpose of estimating fees for L2 transactions (since this also happens based on the local state). This information is necessary to ensure that gas estimations are performed in the exact same manner as the main node, thereby reducing the chances of a transaction not being included in a block.

#

State Keeper / VM

The State Keeper component serves as the "sequencer" part of the node. It shares most of its functionality with the main node, with one key distinction. The main node retrieves transactions from the mempool and has the authority to decide when a specific L2 block or L1 batch should be sealed. On the other hand, the EN retrieves transactions from the queue populated by the Fetcher and seals the corresponding blocks/batches based on the data obtained from the Fetcher queue.

The actual execution of batches takes place within the VM, which is identical in both the Main and External nodes.

#

Reorg Detector

In zkSync Era, it is theoretically possible for L1 batches to be reverted before the corresponding "execute" operation is applied on L1, that is before the block isfinalopen in new window . Such situations are highly uncommon and typically occur due to significant issues: e.g. a bug in the sequencer implementation preventing L1 batch commitment. Prior to batch finality, the zkSync operator can perform a rollback, reverting one or more batches and restoring the blockchain state to a previous point. Finalized batches cannot be reverted at all.

However, even though such situations are rare, the EN must handle them correctly.

To address this, the EN incorporates a Reorg Detector component. This module keeps track of all L1 batches that have not yet been finalized. It compares the locally obtained state root hashes with those provided by the main node's API. If the root hashes for the latest available L1 batch do not match, the Reorg Detector searches for the specific L1 batch responsible for the divergence. Subsequently, it rolls back the local state and restarts the node. Upon restart, the EN resumes normal operation.

#

Consistency Checker

The main node API serves as the primary source of information for the EN. However, relying solely on the API may not provide sufficient security since the API data could potentially be incorrect due to various reasons. The primary source of truth for the rollup system is the L1 smart contract. Therefore, to enhance the security of the EN, each L1 batch undergoes cross-checking against the L1 smart contract by a component called the Consistency Checker.

When the Consistency Checker detects that a particular batch has been sent to L1, it recalculates a portion of the input known as the "block commitment" for the L1 transaction. The block commitment contains crucial data such as the state root and batch number, and is the same commitment that is used for generating a proof for the batch. The Consistency Checker then compares the locally obtained commitment with the actual commitment sent to L1. If the data does not match, it indicates a potential bug in either the main node or external node implementation or that the main node API has provided incorrect data. In either case, the state of the EN cannot be trusted, and the EN enters a crash loop until the issue is resolved.

#

Health check server

The EN also exposes an additional server that returns HTTP 200 response when the EN is operating normally, and HTTP 503 response when some of the health checks don't pass (e.g. when the EN is not fully initialized yet). This server can be used, for example, to implement the readiness probe in an orchestration solution you use.

[ ]Edit this pageopen in new windowLast update: Contributors: [[Peter Straus ,][albicodes]]

Prev Introduction Next Configuration