Bridging with the Espresso Network

Using Espresso confirmations for bridging

Chains that use the Espresso Network to secure their sequencer confirmations benefit from enhanced security when interfacing with external systems, such as bridges, exchanges, and other rollups. This page describes how to use a **caffeinated node** to quickly and securely derive rollup state from data confirmed by Espresso. A caffeinated node can in turn be used:

- by solvers, to quickly release liquidity on behalf of users when bridging out of this change, with minimized risk
- by message passing bridges, to deliver a message on a destination chain quickly without requiring the destination chain to trust the source chain's sequencer
- by exchanges, to quickly and safely fill deposits from the caffeinated chain

Since a caffeinated node is just a modified version of the normal rollup full node, interacting with one once you have it running is easy: just use the usual RPC interface. Thus it should be possible to plug a caffeinated node into your existing solver, bridging protocol, or exchange just by updating your RPC URL.

Using a Caffeinated Node on a Supported Stack

Espresso Systems is developing caffeinated versions of the node software for each stack which we are developing an Espresso integration for. These caffeinated nodes are provided as Docker images, which can be run in the exact same way as a normal full node for each stack, with a few extra parameters for connecting to Espresso. Detailed documentation on running each node will be provided with each release.

The latest list of supported stacks is:

- Arbitrum Nitro (expected early April 2025)
- OP stack (expected Q2 2025)

Writing a Caffeinated Node for a New Rollup Stack

If Espresso Systems has not yet developed an integration for your rollup stack, it is entirely possible to develop your own. All the tech and endpoints are open source. This section gives a brief outline for approaching caffeinated node development.

The basic idea is to modify your full node to read inputs from Espresso, rather than reading inputs from an inbox contract on the L1 or from a sequencer feed. As such, the integration will primarily change how data is ingested. It should not require any changes to the core state transition function of the rollup.

You will need a connection to a server running the Espresso API. Espresso Systems maintains publicly accessible endpoints for mainnet and our Decaf testnet.

The basic flow for ingesting data from Espresso is this, replacing whatever your rollup currently does to read data from a sequencer or inbox contract, and resulting in a stream of blocks which can be executed via the normal state transition function:

- Read the next header from Espresso. This can be done in streaming or polling fashion.

- Verify that this header is finalized. This is done in two parts:
    - by verifying a *light client* proof for HotShot consensus, to verify that the current header *or a later one* is finalized. The consensus light client is currently in development, so existing integrations read a finalized header from the light client contract on the base layer. Espresso Systems currently operates light client contracts on Ethereum and Arbitrum One as well as testnets.
    - by verifying a Merkle proof relative to the known finalized header from the previous step proving the inclusion of the desired header in the Espresso history. This is only necessary if the previous step verified a *later* header. The Merkle proof can be fetched from an untrusted query service via the block-state API.
- Retrieve namespace data from Espresso. This gives you the data which was confirmed for your specific rollup in the finalized Espresso block, along with a proof of Espresso confirmation.
- Verify the namespace proof. This is done by checking a VID proof. The open source Espresso codebase includes Rust and Go libraries for verifying these proofs.
- Namespace verification terminates in a list of Espresso transactions for your rollup's namespace. The caffeinated node must verify that each of these transactions is from an authorized sequencer, or else ignore them, since posting data to Espresso for a given namespace is permissionless. In most rollups with a centralized sequencer, this just requires checking a signature, which should be included with the transaction in the data sent to Espresso.
- Each Espresso transaction is a byte string. In a typical rollup integration, these byte strings encode batches or blocks. From here, you will decode these bytes into your rollup-specific data structure, and execute them exactly as you would had you read data from a sequencer feed or inbox contract.