Introduction to Arbitrum rollups with Celestia as DA

Overview

Theintegration of Celestia with Arbitrum Orbit and the Nitro tech stack marks the first external contribution to the Arbitrum Orbit protocol layer, offering developers an additional option for selecting a data availability layer alongside Arbitrum AnyTrust. The integration allows developers to deploy an Orbit Chain that uses Celestia for data availability and settles on Arbitrum One, Ethereum, or other EVM chains.

Learn more about Orbit in Arbitrum's introduction.

Key components

The integration of Celestia with Arbitrum orbit is possible thanks to 3 key components:

- Introduction to Arbitrum rollups with Celestia as DA
-
    - Overview
-
    - Key components
-
    -
        - DA provider implementation
-
    -
        - Preimage Oracle Implementation
-
    -
        - Blobstream X implementation
-
    -
        - Ethereum fallback mechanism in Nitro
-
    - Next steps

Additionally, theEthereum fallback mechanism is a feature of the integration, which is native in Nitro.

DA provider implementation

The Arbitrum Nitro code has aDataAvailabilityProvider interface that is used across the codebase to store and retrieve data from a specific provider (eip4844 blobs, Anytrust, and now Celestia).

This integration implements theDataAvailabilityProvider interface for Celestia DA

Additionally, this integration comes withthe necessary code for a Nitro chain node to post and retrieve data from Celestia.

The core logic behind posting and retrieving data happens incelestia.go where data is stored on Celestia and serialized into a small batch of data that gets published once the necessary range of headers (data roots) has been relayed to theBlobstreamX contract . Then theRead logic takes care of taking the deserialized Blob Pointer struct and consuming it in order to fetch the data from Celestia and additionally inform the fetcher about the position of the data on Celestia (we'll get back to this in the next section).

The following represents a non-exhaustive list of considerations when running a Batch Poster node for a chain with Celestia underneath:

- You will need to use a consensus node RPC endpoint, you canfind a list of them for Mocha
- The Batch Poster will only post a Celestia batch to the underlying chain if the height for which it posted is in a recent range in BlobstreamX and if the verification succeeds, otherwise it will discard the batch. Since it will wait until a range is relayed, it can take several minutes for a batch to be posted, but one can always make an on-chain request for the BlobstreamX contract to relay a header promptly.

The following represents a non-exhaustive list of considerations when running a Nitro node for a chain with Celestia underneath:

- TheTendermintRpc
- endpoint is only needed by the batch poster, every other node can operate without a connection to a full node.
- The message header flag for Celestia batches is0x0c
- .

- You will need to know the namespace for the chain that you are trying to connect to, but don't worry if you don't find it, as the information in the BlobPointer can be used to identify where a batch of data is in the Celestia Data Square for a given height, and thus can be used to find out the namespace as well!

Preimage Oracle Implementation

In order to support fraud proofs, this integration has the necessary code for a Nitro validator to populate its preimage mapping with Celestia hashes that then get "unpeeled" in order to reveal the full data for a Blob. You canread more about the "Hash Oracle Trick" .

The data structures and hashing functions for this can be found in thenitro/das/celestia/tree folder

You can see where the preimage oracle gets used in the fraud proof replay binaryhere

Something important to note is that the preimage oracle only keeps track of hashes for the rows in the Celestia data square in which a blob resides in, this way each Orbit chain with Celestia underneath does not need validators to recompute an entire Celestia Data Square, but instead, only have to compute the row roots for the rows in which it's data lives in, and the header data root, which is the binary merkle tree hash built using the row roots and column roots fetched from a Celestia node. Because only data roots that can be confirmed on Blobstream get accepted into the sequencer inbox, one can have a high degree of certainty that the canonical data root being unpeeled as well as the row roots are in fact correct.

Blobstream X implementation

Finally, the integration only accepts batches with information that can be confirmed on BlobstreamX, which gives us a high certainty that data was made available on Celestia.

You can see how BlobstreamX is integrated into theSequencerInbox.sol contracthere , which allows us to discard batches with otherwise faulty data roots, thus giving us a high degree of confidence that the data root can be safely unpacked in case of a challenge.

The Celestia and Arbitrum integration alsoincludes Blobstream , which relays commitments to Celestia's data root to an onchain light client on Ethereum. This allows L2 solutions that settle on Ethereum to benefit from the scalability Celestia's data availability layer can provide.

Ethereum fallback mechanism in Nitro

By default inArbitrum Nitro , theEthereum fallback mechanism in theBatchPoster functionis handling the process of storing data, with a fallback mechanism to store data onchain if the primary data availability storage fails.

The@celestiaorg/nitro integrationuses the same fallback mechanism .

More information can be found on the Ethereum fallback mechanisms for Celestia, which enables Ethereum L2s (or L3s) to "fall back" to using Ethereum calldata for data availability in the event of downtime on Celestia Mainnet Beta.

The fallback logic for Celestia DA is configurable, providing an alternative to the previous default fallback mechanism. Additionally, an ability has been added to the Arbitrum node software which allows the sequencer to callVerifyAttestation to check if a data root has been posted on Blobstream or not, before it sends the sequencer message (data pointer) to the underlying chain.

Next steps

In the next page,learn how to deploy an Arbitrum rollup devnet using Celestia as DA. [][Edit this page on GitHub] Last updated: Previous page Overview Next page Quickstart: Deploy an Arbitrum Orbit rollup []