Setting up a Celestia bridge node

This tutorial will go over the steps to setting up your Celestia bridge node.

Bridge nodes connect the data availability layer and the consensus layer.

Overview of bridge nodes

A Celestia bridge node has the following properties:

1. Import and process “raw” headers & blocks from a trusted core process (meaning a trusted RPC connection to a celestia-core node) in the Consensus network. Bridge nodes can run this core process internally (embedded) or simply connect to a remote endpoint. Bridge nodes also have the option of being an active validator in the consensus network.
2. Validate and erasure code the “raw” blocks
3. Supply block shares with data availability headers to light nodes in the DA network.

From an implementation perspective, Bridge nodes run two separate processes:

1. celestia-app with celestia-core (see repo
2.)
3.
    - celestia-app
4.
    - is the state machine where the application and the proof-of-stake logic is run. celestia-app is built onCosmos SDK
5.
    - and also encompassescelestia-core
6.
    - .
7.
    - celestia-core
8.
    - is the state interaction, consensus and block production layer. celestia-core is built onTendermint Core
9.
    - , modified to store data roots of erasure coded blocks among other changes (see ADRs
10.
    -).
11. celestia-node (see repo
12.)
13.
    - celestia-node
14.
    - augments the above with a separate libp2p network that serves data availability sampling requests. The team sometimes refers to this as the “halo” network.

Hardware requirements

The following hardware minimum requirements are recommended for running the bridge node:

- Memory:4 GB RAM (minimum)
- CPU:6 cores
- Disk:10 TB SSD Storage
- Bandwidth:1 Gbps for Download/1 Gbps for Upload

Setting up your bridge node

The following tutorial is done on an Ubuntu Linux 20.04 (LTS) x64 instance machine.

Setup the dependencies

Follow the tutorial forinstalling the dependencies .

Deploy the Celestia bridge node

Install Celestia Node

Install thecelestia-node binary, which will be used to run the bridge node.

Follow the tutorial forinstallingcelestia-node .

Initialize the bridge node

Run the following:

sh celestia

bridge

init

--core.ip

< UR I

celestia

bridge

init

--core.ip

< UR I

The--core.ip gRPC port defaults to 9090, so if you do not specify it in the command line, it will default to that port. You can add the port after the IP address or use the--core.grpc.port flag to specify another port if you prefer.

Refer tothe ports section of the celestia-node troubleshooting page for information on which ports are required to be open on your machine.

Using an RPC of your own, or one from thelist on the Mocha testnet page orlist on the Arabica devnet page , start your node.

Here is an example of initializing the bridge node:

Mainnet Beta

Mocha

Arabica sh celestia

bridge

init

--core.ip

< UR I

celestia

bridge

init

--core.ip

< UR I

sh celestia

bridge

init

--core.ip

< UR I

--p2p.network

mocha celestia

bridge

init

--core.ip

< UR I

--p2p.network

mocha sh celestia

bridge

init

--core.ip

< UR I

--p2p.network

arabica celestia

bridge

init

--core.ip

< UR I

--p2p.network

arabica

Run the bridge node

Start the bridge node with a connection to a validator node's gRPC endpoint (which is usually exposed on port 9090):

sh celestia

bridge

start

--core.ip

< UR I

celestia

bridge

start

--core.ip

< UR I

Here is an example of starting the bridge node on Mocha:

sh celestia

bridge

start

--core.ip

rpc-mocha.pops.one:26657

--p2p.network

mocha celestia

bridge

start

--core.ip

rpc-mocha.pops.one:26657

--p2p.network

mocha And on Arabica:

sh celestia

bridge

start

--core.ip

validator-1.celestia-arabica-11.com

\ --p2p.network

arabica celestia

bridge

start

--core.ip

validator-1.celestia-arabica-11.com

\ --p2p.network

arabica You can create your key for your node by[following thecel-key instructions](#) .

Once you start the bridge node, a wallet key will be generated for you. You will need to fund that address with Testnet tokens to pay forPayForBlob transactions. You can find the address by running the following command:

sh ./cel-key

list

--node.type

bridge

--keyring-backend

test

--p2p.network

< networ k

./cel-key

list

--node.type

bridge

--keyring-backend

test

--p2p.network

< networ k

TIP

You do not need to declare a network for Mainnet Beta. Refer to the chain ID section on the troubleshooting page for more information You can get testnet tokens from:

- Mocha
- Arabica

NOTE

If you are running a bridge node for your validator it is highly recommended to request Mocha testnet tokens as this is the testnet used to test out validator operations.

Optional: run the bridge node with a custom key

In order to run a bridge node using a custom key:

1. The custom key must exist inside the celestia bridge node directory at the correct path (default:~/.celestia-bridge/keys/keyring-test
2.)
3. The name of the custom key must be passed uponstart
4. , like so:

Mainnet Beta

Mocha

Arabica sh celestia

bridge

start

--core.ip

< UR I

--keyring.accname

< name-of-custom-ke y

celestia

bridge

start

--core.ip

< UR I

--keyring.accname

< name-of-custom-ke y

sh celestia

bridge

start

--core.ip

< UR I

--keyring.accname

< name-of-custom-ke y

\ --p2p.network

mocha celestia

bridge

start

--core.ip

< UR I

--keyring.accname

< name-of-custom-ke y

\ --p2p.network

mocha sh celestia

bridge

start

--core.ip

< UR I

--keyring.accname

< name-of-custom-ke y

\ --p2p.network

arabica celestia

bridge

start

--core.ip

< UR I

--keyring.accname

< name-of-custom-ke y

\ --p2p.network

arabica

Optional: start the bridge node with SystemD

Follow thetutorial on setting up the bridge node as a background process with SystemD.

You have successfully set up a bridge node that is syncing with the network. [] Edit this page on GitHub] Last updated:
Previous page Full node Next page Consensus []