Celestia-node RPC CLI tutorial

In this tutorial, we will cover how to use the celestia-node RPC API to submit and retrieve data (blobs) from the data availability layer by their namespace.

Introduction

Blobs

Data is posted to Celestia's DA layer by usingMsgPayForBlobs transactions to the core network. Readmore aboutMsgPayForBlobs .

Namespaces

Celestia partitions the block data into multiple namespaces, one for every application. This allows applications to only download their data, and not the data of other applications. Readmore about Namespaced Merkle trees (NMTs).

TIP

If you already have a running and funded node, you can skip to theRPC CLI guide section .

If you would like to skip syncing, you can use thisguide to sync from trusted hash and height. WARNING

The gateway endpoints have been deprecated and will be removed in the future. If you would like to use them anyway, you canfind more details on GitHub.

Hardware requirements

The following minimum hardware requirements are recommended for running a light node:

- Memory:500 MB RAM (minimum)
- CPU:Single Core
- Disk:50 GB SSD Storage
- Bandwidth:56 Kbps for Download/56 Kbps for Upload

Setting up dependencies

Installdependencies andcelestia-node if you have not already.

Instantiate a Celestia light node

Now, let's instantiate a Celestia Light node:

TIP

RPC endpoints are exposed in all celestia-node types such as light, bridge and full nodes. Mainnet Beta

Mocha

Arabica bash celestia

light

init celestia

light

init bash celestia

light

init

--p2p.network

mocha celestia

light

init

--p2p.network

mocha bash celestia

light

init

--p2p.network

arabica celestia

light

init

--p2p.network

arabica Instantiating (or initializing) the node means setting up a node store on your machine. This is where the data and your keys will be stored.

Connect to a core endpoint

Let's now run the Celestia Light node with a gRPC connection to an example core endpoint. Connecting to a core endpoint provides the light node with access to state queries (reading balances, submitting transactions, and other state-related queries).

Note: You are also encouraged to find an RPC endpoint foMainnet Beta ,Mocha testnet , orArabica devnet . If you are running a production application, use a production endpoint.

Mainnet Beta

Mocha

Arabica bash celestia

light

start

--core.ip

< UR I

    celestia

light

start

--core.ip

< UR I

    bash celestia

light

start

--core.ip

< UR I

--p2p.network

mocha celestia

light

start

--core.ip

< UR I

--p2p.network

mocha bash celestia

light

start

--core.ip

< UR I

--p2p.network

arabica celestia

light

start

--core.ip

< UR I

--p2p.network

arabica TIP

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 to[the ports section of the celestia-node troubleshooting page] for information on which ports are required to be open on your machine. For example, your command along with an RPC endpoint might look like this:

Mainnet Beta

Mocha

Arabica bash celestia

light

start

--core.ip

consensus.lunaroasis.net celestia

light

start

--core.ip

consensus.lunaroasis.net bash celestia

light

start

--core.ip

rpc-mocha.pops.one

--p2p.network

mocha celestia

light

start

--core.ip

rpc-mocha.pops.one

--p2p.network

mocha bash celestia

light

start

--core.ip

validator-1.celestia-arabica-11.com

\ --p2p.network

arabica celestia

light

start

--core.ip

validator-1.celestia-arabica-11.com

\ --p2p.network

arabica

Keys and wallets

You can create your key for your node by running the following command from the celestia-node directory:

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 bash ./cel-key

add

< key-nam e

--keyring-backend

test

--node.type

light

\ --p2p.network

< networ k

    ./cel-key

add

< key-nam e

--keyring-backend

test

--node.type

light

\ --p2p.network

< networ k

You can start your light node with the key created by running the following command:

Mainnet Beta

Mocha

Arabica bash celestia

light

start

--core.ip

< UR I

--keyring.keyname

< key-nam e

celestia

light

start

--core.ip

< UR I

--keyring.keyname

< key-nam e

bash celestia

light

start

--core.ip

< UR I

--keyring.keyname

< key-nam e

\ --p2p.network

mocha celestia

light

start

--core.ip

< UR I

--keyring.keyname

< key-nam e

\ --p2p.network

mocha bash celestia

light

start

--core.ip

< UR I

--keyring.keyname

< key-nam e

\ --p2p.network

arabica celestia

light

start

--core.ip

< UR I

--keyring.keyname

< key-nam e

\ --p2p.network

arabica Once you start the light node, a wallet key will be generated for you. You will need to fund that address with Mocha testnet or Arabica devnet tokens to pay forPayForBlobs transactions.

You can find the address by running the following command in the celestia-node directory:

bash ./cel-key

list

--node.type

light

--keyring-backend

test

--p2p.network

< networ k

    ./cel-key

list

--node.type

light

--keyring-backend

test

--p2p.network

< networ k

> If you would like to fund your wallet with testnet tokens, head over to either the#mocha-faucet or#arabica-faucet channels on theCelestia Discord .

You can request funds to your wallet address using the following command in Discord:

text request request Where is thecelestia1** address generated when you created the wallet.

With your wallet funded, you can move on to the next step.

RPC CLI guide

This section of the tutorial will teach you how to interact with a Celestia node's remote procedure call (RPC) API using the command line interface (CLI).

You will need to setup dependencies, install, and run celestia-node if you have not already.

Command formatting

The format for interacting with the RPC CLI methods is as follows:

bash celestia

< modul e

< metho d

[args...] [flags...] celestia

< modul e

< metho d

[args...] [flags...] Where:

- celestia
- is the main command to interact with the node.
-
- is the specific module in the node you want to interact with, such as blob
- , state
- , p2p
- , etc.
-
- is the specific method within the module that performs the action you want, such as blob.Submit
- , state.AccountAddress
- , p2p.Info
- , etc.
- [args...]
- represents any additional arguments that the method might require.
- [flags...]
- are parameters that modify the behavior of the command. They start with--
- (e.g.,--node.store
- ,--token
- , or--url
-).

For example, to submit a blob to Celestia, you can use this command once your node store is set:

:::note Previously, thenode.store flag had to be specified manually for each request. This has changed in v0.14.0+ and you can read more about the implementation in celestia-node troubleshooting. :::

bash celestia

blob

submit

0x42690c204d39600fddd3

'gm' celestia

blob

submit

0x42690c204d39600fddd3

'gm' TIP

Learn more about maximum blob size. Alternatively, you could use the--token flag to set your auth token:

bash celestia

blob

submit

0x42690c204d39600fddd3

'gm'

--token AUTH_TOKEN celestia

blob

submit

0x42690c204d39600fddd3

'gm'

--token AUTH_TOKEN Before you try that out, let's go over the basic flags that you will need to use when interacting with the RPC CLI. We'll also cover how to set your auth token and how to use the node store to set it.

Basic flags

All RPC CLI commands have basic flags that can be used to interact with the API; however, none are necessary using default configurations.

These include:

- --node.store string
-
    - the path to root/home directory of your celestia-node store
- --token string
-
    - authorization token for making requests
- --url string
-
    - the address of the RPC, default ishttp://localhost:26658

When running RPC CLI commands, you will need to set either theauthentication token or set thenode store , so the auth token can be retrieved from the store.

The RPC CLI handles these flags in the following order:

1. If user passes auth token, auth token is used.
2. If user doesn't pass auth token, check node store flag, create token from node store, and use auth token from node store.

Auth token

In order to interact with the API using RPC CLI, you will need to set the authentication token.

The--token string flag sets the authentication token. If a token is not found, authentication will not be set. And if authentication is not set, the request will fail.

To set your authentication token, you can use the following command. Be sure to replacewith the type of node andwith the network that you are running your node on:

bash export AUTH_TOKEN = (celestia

< node-type

    auth admin --p2p.network

< network

) export AUTH_TOKEN = (celestia

< node-type

    auth admin --p2p.network

< network

) Here's an example of how to set your auth token on a light node on Arabica:

bash export AUTH_TOKEN = (celestia light auth admin --p2p.network arabica) export AUTH_TOKEN = (celestia light auth admin --p2p.network arabica)

Node store

In order to interact with the API using RPC CLI, you can also use your node store to set your auth token. This will allow you to interact with the API without setting an authentication token directly. This is only required if you are using a non-default node store path.

To set a custom node store for a light node on mocha-4, you can use the following command:

bash export NODE_STORE = HOME /your-custom-path/celestia-light-mocha-4 export NODE_STORE = HOME /your-custom-path/celestia-light-mocha-4 Then, set the--node.store flag to theNODE_STORE variable to set the auth token from your node store:

bash celestia

< modul e

< metho d

    [args...] --node.store NODE_STORE celestia

< modul e

< metho d

    [args...] --node.store NODE_STORE

Auth token on custom or private network

This section is for users who are using aCELESTIA_CUSTOM or private network.

TIP

If you are using a private and custom network with a custom node store path, you willneed to set the location of the node store in your auth command. bash --node.store HOME /your-custom-path/.celestia-light-private --node.store HOME /your-custom-path/.celestia-light-private The above is an example from the following custom network set up with:

bash CELESTIA_CUSTOM = private

celestia

light

init CELESTIA_CUSTOM = private

celestia

light

init or

bash celestia

light

init

--p2p.network

private celestia

light

init

--p2p.network

private As an example, this is what a completely custom network would look like:

bash

Initialize node store

CELESTIA_CUSTOM

robusta-22

celestia

light

init

Set auth token

export AUTH_TOKEN = (celestia light auth admin --p2p.network private \ --node.store

HOME /your-custom-path/.celestia-light-robusta-22)

Initialize node store

CELESTIA_CUSTOM

robusta-22

celestia

light

init

Set auth token

export AUTH_TOKEN = (celestia light auth admin --p2p.network private \ --node.store

HOME /your-custom-path/.celestia-light-robusta-22)

Submitting data

In this example, we will be submitting a blob to the network with a blob.Submit transaction with our light node.

Some things to consider:

- The endpoint takes innamespace
- anddata
- values.* The commitment will be generated by the node.
-
    - Share version is set by the node.
- Namespace should be 10 bytes, prefixed by0x
- if hex; otherwise use base64
- Data can be hex-encoded (0x...
-), base64-encoded ("..."
-), or a plaintext string which will be encoded to base64 ('Hello There!'
-)
- Optionally, user can provide a gas fee and gas limit.

We use the followingnamespace of0x42690c204d39600fddd3 and thedata value of0x676d .

Here is an example of the format of theblob.Submit transaction:

bash celestia

blob

submit

< hex-encoded

namespac e

< hex-encoded

dat a

\ [optional: fee] [optional: gasLimit] [optional: node

store

|

auth

token] celestia

blob

submit

< hex-encoded

namespac e

< hex-encoded

dat a

\ [optional: fee] [optional: gasLimit] [optional: node

store

|

auth

token] We run the following to submit a blob to the network in hexadecimal format:

bash celestia

blob

submit

0x42690c204d39600fddd3

0x676d

\ celestia

blob

submit

0x42690c204d39600fddd3

0x676d

\ We get the following output:

json { "result" : { "height" : 252607 , "commitment" : "0MFhYKQUi2BU+U1jxPzG7QY2BVV1Ib3kiU+zAK7nUiY=" } } { "result" : { "height" : 252607 , "commitment" : "0MFhYKQUi2BU+U1jxPzG7QY2BVV1Ib3kiU+zAK7nUiY=" } } We can also use a string of text as the data value, which will be converted to base64. Here is an example of the format:

bash celestia

blob

submit

< hex-encoded

namespac e

< 'data'

\ [optional: fee] [optional: gasLimit] [node store

|

auth

token] celestia

blob

submit

< hex-encoded

namespac e

< 'data'

\ [optional: fee] [optional: gasLimit] [node store

|

auth

token] And an example to submit "gm" as the plain-text data:

bash celestia

blob

submit

0x42690c204d39600fddd3

'gm' celestia

blob

submit

0x42690c204d39600fddd3

'gm' Output:

json { "result" : { "height" : 252614 , "commitment" : "IXg+08HV5RsPF3LIe8PH+B2TUGsGUsBiseflxh6wB5E=" } } { "result" : { "height" : 252614 , "commitment" : "IXg+08HV5RsPF3LIe8PH+B2TUGsGUsBiseflxh6wB5E=" } } If you notice from the above output, it returns aresult of252614 which we will use for the next command. Theresult corresponds to the height of the block in which the transaction was included.

Optional: Submit with curl

Refer to the[submitting a blob using curl](#) section.

Retrieving data

After submitting your PFB transaction, upon success, the node will return the block height for which the PFB transaction was included. You can then use that block height and the namespace ID with which you submitted your PFB transaction to get your message shares (data) returned to you. In this example, the block height we got was 252614 which we will use for the following command. Read more about shares in the[Celestia Specs](#) .

Here is what an example of the format of theget command looks like:

bash celestia

blob

get

< block

heigh t

< hex-encoded

namespac e

\ < commitment

from

output

abov e

< node

store

|

auth> celestia

blob

get

< block

heigh t

< hex-encoded

namespac e

\ < commitment

from

output

abov e

< node

store

|

auth> Here is an example command to retrieve the data from above, onarabica-11 :

bash celestia

blob

get

252614

0x42690c204d39600fddd3

IXg+08HV5RsPF3LIe8PH+B2TUGsGUsBiseflxh6wB5E= celestia

blob

get

252614

0x42690c204d39600fddd3

IXg+08HV5RsPF3LIe8PH+B2TUGsGUsBiseflxh6wB5E= Will generate the following output:

json { "result" : { "namespace" : "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEJpDCBNOWAP3dM=" , "data" : "gm" , "share_version" : 0 , "commitment" : "IXg+08HV5RsPF3LIe8PH+B2TUGsGUsBiseflxh6wB5E=" } } { "result" : { "namespace"

: "AAAAAAAAAAAAAAAAAAAAAAAAAAAEJpDCBNOWAP3dM=" , "data" : "gm" , "share_version" : 0 , "commitment" : "IXg+08HV5RsPF3LIe8PH+B2TUGsGUsBisefIxh6wB5E=" } } The output here is base64 decoded to plain-text.

To see the base64 response, use the--base64 flag set toTRUE (--base64=TRUE):

bash celestia

blob

get

252614

0x42690c204d39600fddd3

IXg+08HV5RsPF3LIe8PH+B2TUGsGUsBisefIxh6wB5E=

\ --base64=TRUE celestia

blob

get

252614

0x42690c204d39600fddd3

IXg+08HV5RsPF3LIe8PH+B2TUGsGUsBisefIxh6wB5E=

\ --base64=TRUE The response will look similar to this:

json { "result" : { "namespace" : "AAAAAAAAAAAAAAAAAAAAAAAAAAAEJpDCBNOWAP3dM=" , "data" : "Z20=" , "share_version" : 0 , "commitment" : "IXg+08HV5RsPF3LIe8PH+B2TUGsGUsBisefIxh6wB5E=" } } { "result" : { "namespace" : "AAAAAAAAAAAAAAAAAAAAAAAAAAAEJpDCBNOWAP3dM=" , "data" : "Z20=" , "share_version" : 0 , "commitment" : "IXg+08HV5RsPF3LIe8PH+B2TUGsGUsBisefIxh6wB5E=" } } To get all blobs in the namespace at the block height, useget-all instead ofget :

bash celestia

blob

get-all

252614

0x42690c204d39600fddd3 celestia

blob

get-all

252614

0x42690c204d39600fddd3 This will return the following:

json { "result" : [{ "namespace" : "AAAAAAAAAAAAAAAAAAAAAAAAAAAEJpDCBNOWAP3dM=" , "data" : "gm" , "share_version" : 0 , "commitment" : "IXg+08HV5RsPF3LIe8PH+B2TUGsGUsBisefIxh6wB5E=" }] } { "result" : [{ "namespace" : "AAAAAAAAAAAAAAAAAAAAAAAAAAAEJpDCBNOWAP3dM=" , "data" : "gm" , "share_version" : 0 , "commitment" : "IXg+08HV5RsPF3LIe8PH+B2TUGsGUsBisefIxh6wB5E=" }] } To display the response in base64, use:

bash celestia

blob

get-all

252614

0x42690c204d39600fddd3

\ --base64=TRUE celestia

blob

get-all

252614

0x42690c204d39600fddd3

\ --base64=TRUE Which will return:

json { "result" : [{ "namespace" : "AAAAAAAAAAAAAAAAAAAAAAAAAAEJpDCBNOWAP3dM=" , "data" : "gm" , "share_version" : 0 , "commitment" : "IXg+08HV5RsPF3LIe8PH+B2TUGsGUsBiseflxh6wB5E=" }] } { "result" : [{ "namespace" : "AAAAAAAAAAAAAAAAAAAAAAAAAAEJpDCBNOWAP3dM=" , "data" : "gm" , "share_version" : 0 , "commitment" : "IXg+08HV5RsPF3LIe8PH+B2TUGsGUsBiseflxh6wB5E=" }] }

Setting the gas price

The--gas.price flag allows you to specify the gas price for the submission. If not specified, a default gas price will be used. The gas limit is automatically calculated based on the size of the blob being submitted.

To set the gas price, you can use the--gas.price flag. The gas price will be set to default (0.002) if no value is passed.

Learnmore about gas fees and limits.

To set a higher gas price of 0.004 utia, use the--gas.price 0.004 flag:

bash celestia

blob

submit

0x42690c204d39600fddd3

'gm'

--gas.price

0.004 celestia

blob

submit

0x42690c204d39600fddd3

'gm'

--gas.price

0.004 You will receive the height and commitment of the block in which the transaction was included for these three examples:

json { "result" : { "height" : 62562 , "commitment" : "IXg+08HV5RsPF3LIe8PH+B2TUGsGUsBiseflxh6wB5E=" } } { "result" : { "height" : 62562 , "commitment" : "IXg+08HV5RsPF3LIe8PH+B2TUGsGUsBiseflxh6wB5E=" } }

Examples

Check your balance

Let's query our node for the balance of its default account (which is the account associated with theCELESTIA_NODE_AUTH_TOKEN key we generated above):

bash celestia

state

balance celestia

state

balance The response will look similar to:

json { "jsonrpc" : "2.0" , "result" : { "denom" : "utia" , "amount" : "172118057" }, "id" : 1 } { "jsonrpc" : "2.0" , "result" : {

"denom" : "utia" , "amount" : "172118057" }, "id" : 1 }

Check the balance of another address

Here is an example of the format of thebalance-for-address command:

bash celestia

state

balance-for-address

< addres s

celestia

state

balance-for-address

< addres s

Let's query our node for the balance of another address:

bash celestia

state

balance-for-address

celestia10rtd9lhel2cuh6c659l25yncl6atcyt37umard celestia

state

balance-for-address

celestia10rtd9lhel2cuh6c659l25yncl6atcyt37umard The response will be the balance of the address you queried:

json { "jsonrpc" : "2.0" , "result" : { "denom" : "utia" , "amount" : "1000000" }, "id" : 1 } { "jsonrpc" : "2.0" , "result" : { "denom" : "utia" , "amount" : "1000000" }, "id" : 1 }

Get your node ID

This is an RPC call in order to get your node's peerId information:

bash celestia

p2p

info celestia

p2p

info The node ID is in theID value from the response:

json { "jsonrpc" : "2.0" , "result" : { "ID" : "12D3KooWFFhCaAqY56oEqY3pLZUdLsv4RYAfVWKATZRepUPdosLp" , "Addrs" : ["/ip4/10.0.0.171/tcp/2121" , "/ip4/10.0.0.171/udp/2121/quic-v1" , "/ip4/71.200.65.106/tcp/25630" , "/ip4/71.200.65.106/udp/25630/quic-v1" , "/ip6/::1/tcp/2121" , "/ip6/::1/udp/2121/quic-v1"] }, "id" : 1 } { "jsonrpc" : "2.0" , "result" : { "ID" : "12D3KooWFFhCaAqY56oEqY3pLZUdLsv4RYAfVWKATZRepUPdosLp" , "Addrs" : ["/ip4/10.0.0.171/tcp/2121" , "/ip4/10.0.0.171/udp/2121/quic-v1" , "/ip4/71.200.65.106/tcp/25630" , "/ip4/71.200.65.106/udp/25630/quic-v1" , "/ip6/::1/tcp/2121" , "/ip6/::1/udp/2121/quic-v1"] }, "id" : 1 }

Get your account address

This is an RPC call in order to get your node's account address:

bash celestia

state

account-address celestia

state

account-address Response:

json { "jsonrpc" : "2.0" , "result" : "celestia1znk24rh52pgcd9z5x2x42jztjh6raaaphuvrt3" , "id" : 1 } { "jsonrpc" : "2.0" , "result" : "celestia1znk24rh52pgcd9z5x2x42jztjh6raaaphuvrt3" , "id" : 1 }

Get block header by height

Here is an example of the format of theGetByHeight command:

bash celestia

header

get-by-height

< heigh t

celestia

header

get-by-height

< heigh t

Now, let's get the block header information.

Here we will get the header from Block 1:

bash celestia

header

get-by-height

1 ```

It will output something like this:

``` json { "jsonrpc" :

"2.0", "result" :

{ "header" :

{ "version" :

{ "block" :

"11", "app" :

"1" }, "chain_id" :

"arabica-11", "height" :

"1", "time" :

"2023-06-27T13:02:39.741743Z", "last_block_id" :

{ "hash" :

"", "parts" :

{ "total" :

0 , "hash" :

"" } }, "last_commit_hash" :

"E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855", "data_hash" :

"3D96B7D238E7E0456F6AF8E7CDF0A67BD6CF9C2089ECB559C659DCAA1F880353", "validators_hash" :
```

"6363C68770C200FD794445668F9B18F5B1DD1125180D6E8D5AB004F7DD7A0F48", "next_validators_hash" :

"6363C68770C200FD794445668F9B18F5B1DD1125180D6E8D5AB004F7DD7A0F48", "consensus_hash" :

"048091BC7DDC283F77BFBF91D73C44DA58C3DF8A9CBC867405D8B7F3DAADA22F", "app_hash" :

"E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855", "last_results_hash" :

"E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855", "evidence_hash" :

"E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855", "proposer_address" :

"91E04695CF9CF531BC0891E7B1D602B3E8022C86" }, "validator_set" :

{ "validators" : [{ "address" :

"91E04695CF9CF531BC0891E7B1D602B3E8022C86", "pub_key" :

{ "type" :

"tendermint/PubKeyEd25519", "value" :

"9aNBAxno1B4X5LR2qY5qWqwrMNOzejkctXwzq9BExsg=" }, "voting_power" :

"500000000", "proposer_priority" :

"0" }], "proposer" :

{ "address" :

"91E04695CF9CF531BC0891E7B1D602B3E8022C86", "pub_key" :

{ "type" :

"tendermint/PubKeyEd25519", "value" :

"9aNBAxno1B4X5LR2qY5qWqwrMNOzejkctXwzq9BExsg=" }, "voting_power" :

"500000000", "proposer_priority" :

"0" } }, "commit" :

{ "height" :

1 , "round" :

0 , "block_id" :

{ "hash" :

"7A5FABB19713D732D967B1DA84FA0DF5E87A7B62302D783F78743E216C1A3550", "parts" :

{ "total" :

1 , "hash" :

"D85C907CE660878A8203AC74BAA147CCC1F87114B45B568B72AD207B62AFE45E" } }, "signatures" : [{ "block_id_flag" :

2 , "validator_address" :

"91E04695CF9CF531BC0891E7B1D602B3E8022C86", "timestamp" :

"2023-06-30T08:40:19.299137127Z", "signature" :

"qmaEzrnbtgEXCRYc8pCvGRbS+uMuknIBoRAE4qyE7oSgWCRwBVYS/oPReXQLg9ER1oEY1De4MkWvMjlFnQOOCg==" }] }, "dah" :

{ "row_roots" : [
"//////////////////////////////////////7//////////////////////////////////////huZWOTTDmD36N1F75A9BshxNlRasCnNpQiWqIhdVHcU" ,
"///5iieeroHBMfF+sER3JpvROIeEJZjbY+TRE0ntADQLL3"], "column_roots" :
["//////////////////////////////////////7//////////////////////////////////////huZWOTTDmD36N1F75A9BshxNlRasCnNpQiWqIhdVHcU" ,
"///5iieeroHBMfF+sER3JpvROIeEJZjbY+TRE0ntADQLL3"] } }, "id" :

1 } celestia

header

get-by-height

1 ```

It will output something like this:

``` json { "jsonrpc" :

"2.0", "result" :

{ "header" :

{ "version" :

{ "block" :

"11", "app" :

"1" }, "chain_id" :

"arabica-11", "height" :

"1", "time" :

"2023-06-27T13:02:39.741743Z", "last_block_id" :

{ "hash" :

"", "parts" :

{ "total" :

0 , "hash" :

"" } }, "last_commit_hash" :

"E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855", "data_hash" :

"3D96B7D238E7E0456F6AF8E7CDF0A67BD6CF9C2089ECB559C659DCAA1F880353", "validators_hash" :

"6363C68770C200FD794445668F9B18F5B1DD1125180D6E8D5AB004F7DD7A0F48", "next_validators_hash" :

"6363C68770C200FD794445668F9B18F5B1DD1125180D6E8D5AB004F7DD7A0F48", "consensus_hash" :

"048091BC7DDC283F77BFBF91D73C44DA58C3DF8A9CBC867405D8B7F3DAADA22F", "app_hash" :

"E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855", "last_results_hash" :

"E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855", "evidence_hash" :

"E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855", "proposer_address" :

"91E04695CF9CF531BC0891E7B1D602B3E8022C86" }, "validator_set" :

{ "validators" : [ { "address" :

"91E04695CF9CF531BC0891E7B1D602B3E8022C86", "pub_key" :

{ "type" :

"tendermint/PubKeyEd25519", "value" :

"9aNBAxno1B4X5LR2qY5qWqwrMNOzejkctXwzq9BExsg=" }, "voting_power" :

"500000000", "proposer_priority" :

"0" } ], "proposer" :

{ "address" :
```

"91E04695CF9CF531BC0891E7B1D602B3E8022C86", "pub_key" :

{ "type" :

"tendermint/PubKeyEd25519", "value" :

"9aNBAxno1B4X5LR2qY5qWqwrMNOzejkctXwzq9BExsg=" }, "voting_power" :

"500000000", "proposer_priority" :

"0" } }, "commit" :

{ "height" :

1 , "round" :

0 , "block_id" :

{ "hash" :

"7A5FABB19713D732D967B1DA84FA0DF5E87A7B62302D783F78743E216C1A3550", "parts" :

{ "total" :

1 , "hash" :

"D85C907CE660878A8203AC74BAA147CCC1F87114B45B568B72AD207B62AFE45E" } }, "signatures" : [{ "block_id_flag" :

2 , "validator_address" :

"91E04695CF9CF531BC0891E7B1D602B3E8022C86", "timestamp" :

"2023-06-30T08:40:19.299137127Z", "signature" :

"qmaEzrnbtgEXCRYc8pCvGRbS+uMuknIBoRAE4qyE7oSgWCRwBVYS/oPReXQLg9ER1oEY1De4MkWvMjlFnQOOCg==" }] }, "dah" :

{ "row_roots" : [
"//////////////////////////////////////7//////////////////////////////////////huZWOTTDmD36N1F75A9BshxNlRasCnNpQiWqIhdVHcU" ,
"///5iieeroHBMfF+sER3JpvROIeEJZjbY+TRE0ntADQLL3"], "column_roots" :
["//////////////////////////////////////7//////////////////////////////////////huZWOTTDmD36N1F75A9BshxNlRasCnNpQiWqIhdVHcU" ,
"///5iieeroHBMfF+sER3JpvROIeEJZjbY+TRE0ntADQLL3"] } }, "id" :

1 }

Combined commands

bash celestia

share

get-by-namespace

"(celestia header get-by-height 147105

|

jq '.result.dah' -r)"

0x42690c204d39600fddd3 celestia

share

get-by-namespace

"(celestia header get-by-height 147105

|

jq '.result.dah' -r)"

0x42690c204d39600fddd3

Get data availability sampler stats

bash celestia

das

sampling-stats celestia

das

sampling-stats

Transfer balance of utia to another account

First, set your address as a variable:

bash export ADDRESS = celestia1c425ckmve2489atttx022qpc02gxspa29wmh0d export ADDRESS = celestia1c425ckmve2489atttx022qpc02gxspa29wmh0d Then, transfer the amount of tokens that you would like, while setting the recipient's address, gas fee, and gasLimit. This is what the format will look like:

bash celestia

state

transfer ADDRESS < amount

in

uti a

< gas

fee

in

uti a

< gas

fee

in

uti a

celestia

state

transfer ADDRESS < amount

in

uti a

< gas

fee

in

uti a

< gas

fee

in

uti a

Here is an example, sending 0.1 TIA, with a gas fee of 0.008 TIA, and a gas limit of 0.08:

bash celestia

state

transfer ADDRESS 100000

8000

80000 celestia

state

transfer ADDRESS 100000

8000

80000 If you'd just like to return the transaction hash, you can use jq:

bash celestia

state

transfer ADDRESS 100000

8000

80000

|

jq

.result.txhash celestia

state

transfer ADDRESS 100000

8000

80000

|

jq

.result.txhash

API version

To query your node's API version, you can use the following command:

bash celestia

node

info celestia

node

info

Help

To get help and view the CLI menu, use the following command:

bash celestia

--help celestia

--help To view the help menu for a specific method, use the following command:

bash celestia

< modul e

< metho d

--help celestia

< modul e

< metho d

--help

Advanced example

This example shows us using thejq command to parse the output of thecelestia header get-by-height method to get the extended header used incelestia share get-by-namespace :

bash celestia

share

get-by-namespace

\ "(celestia header get-by-height 252614

|

jq '.result.dah' -r)"

\ 0x42690c204d39600fddd3 celestia

share

get-by-namespace

\ "(celestia header get-by-height 252614

|

jq '.result.dah' -r)"

\ 0x42690c204d39600fddd3

Additional resources

Submitting a blob using curl

In order to post a blob using curl, you will need a light node running with the--core.ip string flag, providing access to a consensus endpoint. The flag indicates node to connect to the given core consensus node. Examples:127.0.0.1 orsubdomain.domain.tld . Using either IP or DNS assumes RPC port 26657 and gRPC port 9090 as default unless otherwise specified.

1. In your terminal, set the auth token for the desired network. In this example, we will use Mainnet Beta.

bash export CELESTIA_NODE_AUTH_TOKEN = (celestia light auth admin --p2p.network celestia) export CELESTIA_NODE_AUTH_TOKEN = (celestia light auth admin --p2p.network celestia) 1. Post your blob with:

bash curl

-H

"Content-Type: application/json"

-H

"Authorization: Bearer CELESTIA_NODE_AUTH_TOKEN "

-X

POST

--data

'{"id": 1, "jsonrpc": "2.0", "method": "blob.Submit", "params": [[{ "namespace": "AAAAAAAAAAAAAAAAAAAAAAAAAAECAwQFBgcICRA=", "data": "VGhpcyBpcyBhbiBleGFtcGxlIG9mIHNvbWUgYmxvYiBkYXRh", "share_version": 0, "commitment": "AD5EzbG0/EMvpw0p8NIjMVnoCP4Bv6K+V6gjmwdXUKU=" }], 0.002] }'

127.0 .0.1:26658 curl

-H

"Content-Type: application/json"

-H

"Authorization: Bearer CELESTIA_NODE_AUTH_TOKEN "

-X

POST

--data

'{"id": 1, "jsonrpc": "2.0", "method": "blob.Submit", "params": [[{ "namespace": "AAAAAAAAAAAAAAAAAAAAAAAAAAECAwQFBgcICRA=", "data": "VGhpcyBpcyBhbiBleGFtcGxlIG9mIHNvbWUgYmxvYiBkYXRh", "share_version": 0, "commitment": "AD5EzbG0/EMvpw0p8NIjMVnoCP4Bv6K+V6gjmwdXUKU=" }], 0.002] }'

127.0 .0.1:26658 1. Upon successful blob submission, the result will show the block height:

bash { "jsonrpc" : "2.0" , "result" :362101, "id" :1} { "jsonrpc" : "2.0" , "result" :362101, "id" :1} The example transaction can befound on Celenium .

Post an SVG as a PFB

If you'd like to create your own SVG, post it to Celestia, and retrieve it, you can check out theBase64 SVG Tutorial .

Troubleshooting

If you encounter an error like:

sh "rpc error: code = NotFound desc = account celestia1krkle0n547u0znz3unnln8paft2dq4z3rznv86 not found" "rpc error: code = NotFound desc = account celestia1krkle0n547u0znz3unnln8paft2dq4z3rznv86 not found" It is possible that the account you are trying to submit aPayForBlobs from doesn't have testnet tokens yet. Ensure the testnet faucet has funded your account with tokens and then try again. [][Edit this page on GitHub] Last updated: Previous page Overview Next page Golang client tutorial []