Quick-start guide

Welcome to Celestia's quick-start guide! In this guide, we'll learn how to run a Celestia data availability sampling (DAS) light node to post and retrieve data blobs on Celestia'sMocha testnet .

A blob (a.k.a.BLOB) is a Binary Large OBject. In other words, a blob is arbitrary data. In this case, it's data that you want to post and make available on Celestia's data availability (DA) layer.

Your light node will allow you to post data, and then use DAS to sample and retrieve it from the DA network. Let's get started!

Run a light node

First we'll need to install thecelestia binary to run our DAS light node. Use the following command to install a pre-built binary ofcelestia-node , for the latest release for Mocha testnet:

For this guide, select either your Go bin or system bin directory when prompted. If you're curious whatthe script is doing, check outthe celestia-node page . bash bash

-c

"(curl

-sL https://docs.celestia.org/celestia-node.sh)"

--

-v

v0.18.3-mocha bash

-c

"(curl

-sL https://docs.celestia.org/celestia-node.sh)"

--

-v

v0.18.3-mocha Once you've installedcelestia , double-check that you're using the right version by running:

bash celestia

version celestia

version You should see the version of the binary that you just installed. Usecelestia --help to see the CLI menu.

Initialize the light node

Initializing your light node will set up configuration files and create a keypair for your node.

Thep2p.network flag is used to specify the network you want to connect to. Usemocha for Mocha testnet.

The chain IDmocha-4 is also an accepted alias. bash celestia

light

init

--p2p.network

mocha celestia

light

init

--p2p.network

mocha Once you've run this command, you'll see a new keypair that's created in your terminal. Be sure to save your mnemonic somewhere safe for future use!

bash INFO

node

nodebuilder/init.go:31

Initializing

Light

Node

Store

over

'/Users/js/.celestia-light-mocha-4' INFO

node

nodebuilder/init.go:64

Saved

config

{"path":

"/Users/js/.celestia-light-mocha-4/config.toml"} INFO

node

nodebuilder/init.go:66

Accessing

keyring... WARN

node

nodebuilder/init.go:196

Detected

plaintext

keyring

backend.

For

elevated

security

properties,

consider

using

the

file

keyring

backend. INFO

node

nodebuilder/init.go:211

NO

KEY

FOUND

IN

STORE,

GENERATING

NEW

KEY...

{"path":

"/Users/js/.celestia-light-mocha-4/keys"} INFO

node

nodebuilder/init.go:216

NEW

KEY

GENERATED...

NAME:

my_celes_key ADDRESS:

celestia1lgvzg4ek9v499pl5vvsvqpquhwfg0jznpwd92m MNEMONIC (save this

somewhere

safe!!!): never

gonna

give

you

up

never

gonna

let

you

down

never

gonna

run

around

and

desert

you

never

gonna

make

you

cry

never

gonna

INFO

node

nodebuilder/init.go:73

Node

Store

initialized INFO

node

nodebuilder/init.go:31

Initializing

Light

Node

Store

over

'/Users/js/.celestia-light-mocha-4' INFO

node

nodebuilder/init.go:64

Saved

config

{"path":

"/Users/js/.celestia-light-mocha-4/config.toml"} INFO

node

nodebuilder/init.go:66

Accessing

keyring... WARN

node

nodebuilder/init.go:196

Detected

plaintext

keyring

backend.

For

elevated

security

properties,

consider

using

the

file

keyring

backend. INFO

node

nodebuilder/init.go:211

NO

KEY

FOUND

IN

STORE,

GENERATING

NEW

KEY...

{"path":

"/Users/js/.celestia-light-mocha-4/keys"} INFO

node

nodebuilder/init.go:216

NEW

KEY

GENERATED...

NAME:

my_celes_key ADDRESS:

celestia1lgvzg4ek9v499pl5vvsvqpquhwfg0jznpwd92m MNEMONIC (save this

somewhere

safe!!!): never

gonna

give

you

up

never

gonna

let

you

down

never

gonna

run

around

and

desert

you

never

gonna

make

you

cry

never

gonna

INFO

node

nodebuilder/init.go:73

Node

Store

initialized You'll also see in this example, using the Mocha testnet and setting up a light node, our node store will be at:~/.celestia-light-mocha-4 .

Logs above have the timestamps removed for brevity. And yes, that's a Rickroll mnemonic. 😆

Set the trusted hash

Setting and syncing to a trusted hash and height means your light node will not sample the entire chain. This is useful when you want to sync your light node quickly. However, it's important to note that this adds the trust assumption that you trust the entity where you get the hash and height from, in this case, theP-OPS team's consensus endpoint.

Let's set the trusted hash!

1. Get trusted height & hash from the P-OPS consensus endpoint:
2. bash
3. export
4. TRUSTED_HEIGHT
5. =
6. (
7. curl
8. -s
9. "https://rpc-mocha.pops.one/header"
10. |
11. jq
12. -r
13. '.result.header.height')

14. &&
15. export
16. TRUSTED_HASH
17. =
18. (
19. curl
20. -s
21. "https://rpc-mocha.pops.one/header"
22. |
23. jq
24. -r
25. '.result.header.last_block_id.hash')
26. &&
27. echo
28. "Height:
29. TRUSTED_HEIGHT
30. "
31. &&
32. echo
33. "Hash:
34. TRUSTED_HASH
35. "
36. export
37. TRUSTED_HEIGHT
38. =
39. (
40. curl
41. -s
42. "https://rpc-mocha.pops.one/header"
43. |
44. jq
45. -r
46. '.result.header.height')
47. &&
48. export
49. TRUSTED_HASH
50. =
51. (
52. curl
53. -s
54. "https://rpc-mocha.pops.one/header"
55. |
56. jq
57. -r
58. '.result.header.last_block_id.hash')
59. &&
60. echo
61. "Height:
62. TRUSTED_HEIGHT
63. "
64. &&
65. echo
66. "Hash:
67. TRUSTED_HASH
68. "
69. Set the trusted height & hash
70.
    1. Open yourconfig.toml
71.
    1. at.celestia-light-mocha-4/config.toml
72.
    1. SetDASer.SampleFrom
73.
    1. to the trusted height (e.g.SampleFrom = 123456
74.
    1.)

If you dont do this, when trying to retrieve data in a few minutes, you'll see a response saying"result": "header: syncing in progress: localHeadHeight: 94721, requestedHeight: 2983850" . You'll either need to let the node sync to

therequestedHeight , or use quick sync with trusted hash to do this. Learn more inthe trusted hash quick sync guide .

Start the light node

Run the following command to start your light node:

In the same terminal you initialized the node store and set the variable forTRUSTED_HASH , start the node with the hash and flag:

bash celestia

light

start

--headers.trusted-hash TRUSTED_HASH \ --p2p.network

mocha

--core.ip

rpc-mocha.pops.one celestia

light

start

--headers.trusted-hash TRUSTED_HASH \ --p2p.network

mocha

--core.ip

rpc-mocha.pops.one Thecore.ip flag is used to specify the consensus RPC endpoints you want to connect to, this is the same one we got the trusted height and hash from. We'll userpc-mocha.pops.one from the P-OPS team for Mocha testnet. Theheaders.trusted-hash flag will set the trusted hash from the previous section.

Once you see this in the logs, you're ready to start posting and retrieving data!

bash /_____/

/_____/

/_____/

/_____/

/_____/

Started

celestia

DA

node node

version:

0.18.3-mocha node

type:

light network:

mocha-4

/_____/

/_____/

/_____/

/_____/

/_/ /___/

/_____/

/_____/

/_____/

/_____/

Started

celestia

DA

node node

version:

0.18.3-mocha node

type:

light network:

mocha-4

/_____/

/_____/

/_____/

/_____/

/_____/ TIP

If you want to see that your node is synced, use thecelestia das sampling-stats command to check it in another terminal:

bash { "result" :

{ "head_of_sampled_chain" :

2990507 , "head_of_catchup" :

2990507 , "network_head_height" :

2990507 , "concurrency" :

0 , "catch_up_done" :

true , "is_running" :

true } } { "result" :

{ "head_of_sampled_chain" :

2990507 , "head_of_catchup" :

2990507 , "network_head_height" :

2990507 , "concurrency" :

0 , "catch_up_done" :

true , "is_running" :

true } }

Post and retrieve data with your light node

Funding your light node

Now, we're almost ready to start posting data!

Open a second terminal instance for the remainder of this guide. Let's find our address for the node we have running by running:

bash celestia

state

account-address celestia

state

account-address Take this account address and head over to theDiscord and request tokens from the#mocha-faucet channel.

Once you've requested tokens, can check the balance of your running node using:

bash celestia

state

balance celestia

state

balance

Posting data

Now that you have tokens in your account, you can post data to the network. Let's break down the arguments you'll provide to the CLI:

bash celestia

blob

submit [namespace] [blobData] celestia

blob

submit [namespace] [blobData] The[namespace] is a permissionless way to categorize your data on Celestia. In other words, it's a channel for you to post your data. For example, this could be the name of your project or a category for the type of blob. In this example, we'll use0x71756f746573 as the namespace, which is the hex encoding of "quotes". See the"quotes" namespace on Celenium .

Learn more about namespaces inthe celestia-app documentation . The[blobData] is the blob data you want to post to the network. In this example, we'll use a quote from Leonardo da Vinci:

TIP

Feeling creative? Post your favorite quote andshare it on Twitter ! bash celestia

blob

submit

0x71756f746573

'"Simplicity is the ultimate sophistication." -Leonardo da Vinci' celestia

blob

submit

0x71756f746573

'"Simplicity is the ultimate sophistication." -Leonardo da Vinci' Once you run this command, you'll see a height and data committment in the response. This means your data has been successfully posted to the network!

bash { "result" :

{ "height" :

2990556 , "commitments" : ["0x715ab246772c923104c556dc28d5d4fcfca0398b0a252bcd19dd5705495756ac"] } } { "result" :

{ "height" :

2990556 , "commitments" : ["0x715ab246772c923104c556dc28d5d4fcfca0398b0a252bcd19dd5705495756ac"] } }

Retrieving data

To retrieve the data you posted, use the following command:

bash celestia

blob

get [height] [namespace] [committment] celestia

blob

get [height] [namespace] [committment] In this example, you'll use the height and commitment from the response above:

bash celestia

blob

get

2990556

0x71756f746573

0x715ab246772c923104c556dc28d5d4fcfca0398b0a252bcd19dd5705495756ac celestia

blob

get

2990556

0x71756f746573

0x715ab246772c923104c556dc28d5d4fcfca0398b0a252bcd19dd5705495756ac In response, you'll see the data you posted:

bash { "result" :

{ "namespace" :

"0x71756f746573", "data" :

" \" Simplicity is the ultimate sophistication. \" -Leonardo da Vinci", "share_version" :

0 , "commitment" :

"0x715ab246772c923104c556dc28d5d4fcfca0398b0a252bcd19dd5705495756ac", "index" :

29 } } { "result" :

{ "namespace" :

"0x71756f746573", "data" :

" \" Simplicity is the ultimate sophistication. \" -Leonardo da Vinci", "share_version" :

0 , "commitment" :

"0x715ab246772c923104c556dc28d5d4fcfca0398b0a252bcd19dd5705495756ac", "index" :

29 } } Let's break it down:

- namespace
- : Thenamespace

- you used to categorize your data.
- data
- : The data blob you posted to the network.
- share_version
- : The version of theshare
- . A share is a fixed-size data chunk that is associated with exactly one namespace.
- commitment
- : The commitment of the data (seeBlob Share Commitment Rules
-).
- index
- : Theindex
- of the data share in the square.

Congratulations! You've successfully learned how to run a light node to post and retrieve data from Celestia's Mocha testnet.

Diving deeper into the stack

This section covers some more in-depth topics that you may find useful when working with your Celestia light node.

Get your auth token

Your auth token may be useful when you want to interact with your Celestia light node from a client application. You can get your auth token by running:

bash celestia

light

auth

admin

--p2p.network

mocha celestia

light

auth

admin

--p2p.network

mocha Usecelestia light auth --help to learn more about the available options.

Key management with cel-key

In the first part of this guide, we generated a key when we initialized the light node.

An advanced option for key management is using thecel-key utility, which is a separate tool from thecelestia binary.cel-key is a key management tool that allows you to create, import, and manage keys for your Celestia DA node.

If you're using the quickstart script above, you will have to buildcel-key separately from source. You can find the instructions for buildingcel-key in thecelestia-node documentation.

Rust client tutorial

If you're interested in writing a Rust program to interact with your Celestia light node, check out theRust client tutorial .

Golang client tutorial

If you're interested in writing a Golang program to interact with your Celestia light node, check out theGolang client tutorial .

Node store contents

As described in theinitialize the light node section above, the node store is created in the~/.celestia- directory.

In this guide, the node store for~/.celestia-light-mocha-4 contains the following directories and file types:

- config.toml
- : Node configuration settings
- data/
- : Contains database files* .vlog
  - files: Value log files storing actual data
  - .sst
  - files: Static sorted tables containing indexed data
  - System files:DISCARD
  - ,KEYREGISTRY
  - , andMANIFEST
  - for database management
- keys/
- : Stores node identity and account keys* Contains encoded node identifiers
  - keyring-test/
  - : Test keyring directory* .address
    - files: Account addresses
    - .info
    - files: Key metadata and information

Troubleshooting

If you run into issues, check out thetroubleshooting page for common problems and solutions.

Next steps

Check out thebuild whatever page to get started learning about ways to build with Celestia underneath.

Head to the next page to learn about different node types for the consensus and DA networks. [][Edit this page on GitHub]
Last updated: Previous page Overview Next page Deciding which node to run []