Deploy an OP Stack devnet

This guide will show you how to run your own OP Stack devnet.

Dependency setup

Environment setup and Golang installation

First,install dependencies for Celestia software and forOP Stack .

Clone repository

Next, clone the repo:

bash cd HOME git

clone

https://github.com/celestiaorg/optimism cd

optimism cd HOME git

clone

https://github.com/celestiaorg/optimism cd

optimism Check out to the version for either the stable version or upstream version:

v1.2.0-OP_v1.7.0-CN_v0.12.4

rebase-upstream bash git

checkout

tags/v1.2.0-OP_v1.7.0-CN_v0.12.4 git

submodule

update

--init

--recursive git

checkout

tags/v1.2.0-OP_v1.7.0-CN_v0.12.4 git

submodule

update

--init

--recursive bash git

checkout

rebase-upstream git

submodule

update

--init

--recursive git

checkout

rebase-upstream git

submodule

update

--init

--recursive

Build devnet

Build TypeScript definitions for TS dependencies:

bash cd HOME cd

optimism make cd HOME cd

optimism make Set environment variables to start network:

bash export SEQUENCER_BATCH_INBOX_ADDRESS = 0xff0000000000000000000000000000000000000 export L2OO_ADDRESS = 0x70997970C51812dc3A010C7d01b50e0d17dc79C8 export SEQUENCER_BATCH_INBOX_ADDRESS = 0xff0000000000000000000000000000000000000 export L2OO_ADDRESS = 0x70997970C51812dc3A010C7d01b50e0d17dc79C8

Start devnet

Start the network by running:

bash make

devnet-up make

devnet-up This starts up the layer 1 (ETH), layer 2 (op-geth), data availability layer (Celestia), the sequencer (op-node), batch submitter (op-batcher), state commitment service (op-proposer).

View the logs of the devnet

If you'd like to view the logs of the devnet, run the following command from the root of the Optimism directory:

bash make

devnet-logs make

devnet-logs Optional: Docker tips

Stop devnet

If you'd like to start the network over, use the following command to safely shut down all of the containers:

bash make

devnet-down make

devnet-down Then clean out the old config:

bash make

devnet-clean make

devnet-clean

Viewing containers

To view the containers running, send:

bash docker

ps docker

ps Find the container ID of theops-bedrock_op-batcher_1 and run the following to follow the logs:

bash docker

logs

-f

< container-i d

docker

logs

-f

< container-i d

In a new terminal, find the container ID of theghcr.io/celestiaorg/local-celestia-devnet:main and run the following to follow the logs:

bash docker

logs

-f

< container-i d

docker

logs

-f

< container-i d

You can do the same for other containers as you explore the stack.

Find a transaction

Now, we'll check for a recent transaction on the L1 with:

bash cast

block

latest

--rpc-url

localhost:8545 cast

block

latest

--rpc-url

localhost:8545 Output of a block that contains a transaction will look like this:

console baseFeePerGas 7 difficulty 2 extraData
0xd883010d0484676574688867 6f312e32312e33856c696e7578000000000000006b3afa42dce1f87f1f07a1ef569c4d43e41738ef93c865098bfa1458645f384e2e4498bcfe4ad9353ff1913a2e16162f496fafe
gasLimit 30000000 gasUsed 21568 hash 0x1cb54d2369752ef73511c202ff9cdfd0eadf3a77b7aef0092bea63f2b5d57659 logsBloom
0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
miner 0x0000000000000000000000000000000000000000 mixHash 0x0000000000000000000000000000000000000000000000000000000000000000 nonce 0x0000000000000000 number 1141
parentHash 0x664bf4bb4a57dd5768a0a98991d77c58fb7a4e164c2581c79fb33ce9c3d4c250 receiptsRoot 0xaf8ff6af1180c8be9e4e8f3a5f882b3b227233f4abbefa479836d3721682a389 sealFields []
sha3Uncles 0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347 size 767 stateRoot 0xd4b998a35d20d98ed3488221f0c161a0a9572d3de66399482553c8e3d2fae751
timestamp 1699638350 withdrawalsRoot totalDifficulty 2283 transactions: [0x79a0a7a1b4936aafe7a37dbfb07a6a9e55c145a4ed6fd54f962649b4b7db8de7] baseFeePerGas 7 difficulty 2 extraData
0xd883010d0484676574688867 6f312e32312e33856c696e7578000000000000006b3afa42dce1f87f1f07a1ef569c4d43e41738ef93c865098bfa1458645f384e2e4498bcfe4ad9353ff1913a2e16162f496fafe
gasLimit 30000000 gasUsed 21568 hash 0x1cb54d2369752ef73511c202ff9cdfd0eadf3a77b7aef0092bea63f2b5d57659 logsBloom
0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
miner 0x0000000000000000000000000000000000000000 mixHash 0x0000000000000000000000000000000000000000000000000000000000000000 nonce 0x0000000000000000 number 1141
parentHash 0x664bf4bb4a57dd5768a0a98991d77c58fb7a4e164c2581c79fb33ce9c3d4c250 receiptsRoot 0xaf8ff6af1180c8be9e4e8f3a5f882b3b227233f4abbefa479836d3721682a389 sealFields []
sha3Uncles 0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347 size 767 stateRoot 0xd4b998a35d20d98ed3488221f0c161a0a9572d3de66399482553c8e3d2fae751
timestamp 1699638350 withdrawalsRoot totalDifficulty 2283 transactions: [0x79a0a7a1b4936aafe7a37dbfb07a6a9e55c145a4ed6fd54f962649b4b7db8de7] Copy the transaction hash
fromtransactions: and set it as a variable:

bash export TX_HASH = 0x79a0a7a1b4936aafe7a37dbfb07a6a9e55c145a4ed6fd54f962649b4b7db8de7 export TX_HASH =
0x79a0a7a1b4936aafe7a37dbfb07a6a9e55c145a4ed6fd54f962649b4b7db8de7

Read the transaction call data

Now read the transaction call data on the L1:

bash cast

tx TX_HASH --rpc-url

localhost:8545 cast

tx TX_HASH --rpc-url

localhost:8545 The output will look similar to below:

console blockHash 0x9f4dfae061b5ddd86f95a81be5daa0d7fe32e7f7f770f86dc375e0007d249bd2 blockNumber 24 from 0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC gas 21572 gasPrice 1040676758 hash 0xadd3a5dc0b8c605aeac891098e87cbaff43bb642896ebbf74f964c0690e46df2 input 0xce350000000000000000769074a923011bdda721eacc34c8a77c69c10f2b6c8e659f987e82f217a5340f nonce 4 r 0xaf5c1505c7dfcebca94d9a6a8c0caf99b6c87a8ed6d6c0b3161c9026f270a84f s 0x383ed2debf9f9055920cd7340418dda7e2bca6b989eb6992d83d123d4e322f2a to 0xFf0000000000000000000000000000000000000901 transactionIndex 0 v 0 value 0 yParity 0 blockHash 0x9f4dfae061b5ddd86f95a81be5daa0d7fe32e7f7f770f86dc375e0007d249bd2 blockNumber 24 from 0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC gas 21572 gasPrice 1040676758 hash 0xadd3a5dc0b8c605aeac891098e87cbaff43bb642896ebbf74f964c0690e46df2 input 0xce350000000000000000769074a923011bdda721eacc34c8a77c69c10f2b6c8e659f987e82f217a5340f nonce 4 r 0xaf5c1505c7dfcebca94d9a6a8c0caf99b6c87a8ed6d6c0b3161c9026f270a84f s 0x383ed2debf9f9055920cd7340418dda7e2bca6b989eb6992d83d123d4e322f2a to 0xFf0000000000000000000000000000000000000901 transactionIndex 0 v 0 value 0 yParity 0 TIP

You are looking for a batcher transaction to the address0xFf0000000000000000000000000000000000000901 . First, remove the prefix0xce . Now, set theinput as theINPUT variable and encode it as base64:

bash export INPUT = ce350000000000000000769074a923011bdda721eacc34c8a77c69c10f2b6c8e659f987e82f217a5340f export ENCODED_INPUT = (echo " INPUT " |

xxd

-r

-p

|

base64) export INPUT = ce350000000000000000769074a923011bdda721eacc34c8a77c69c10f2b6c8e659f987e82f217a5340f export ENCODED_INPUT = (echo " INPUT " |

xxd

-r

-p

|

base64) TIP

Remember to remove the0xce prefix!

Find the data on Celestia

Clone thego-da repository:

bash cd HOME git

clone

https://github.com/rollkit/go-da.git cd

go-da/proto/da cd HOME git

clone

https://github.com/rollkit/go-da.git cd

go-da/proto/da Now, fromgo-da/proto/da run:

bash grpcurl

-proto

da.proto

-plaintext

-d

"{ \" ids \" : [{ \" value \" : \" ENCODED_INPUT \" }]}"

127.0 .0.1:26650

da.DAService.Get grpcurl

-proto

da.proto

-plaintext

-d

"{ \" ids \" : [{ \" value \" : \" ENCODED_INPUT \" }]}"

127.0 .0.1:26650

da.DAService.Get Your result will look similar to the below!

console { "blobs": [{ "value":
"AKUumhJ8FnuyVrBs38FDKEIAAAAAAZB42trhw/DDc4GFAlv4klkv5Zh4E16mmO5fpNOS1f5wzpds8YK3S0Rvs4ULLJj13euw+Ovdv6Q23zuV1ShROEvk5aptIT7bGmZunvc1OiKwJTXVbN0BiGm6k2zNW
}] } { "blobs": [{ "value":
"AKUumhJ8FnuyVrBs38FDKEIAAAAAAZB42trhw/DDc4GFAlv4klkv5Zh4E16mmO5fpNOS1f5wzpds8YK3S0Rvs4ULLJj13euw+Ovdv6Q23zuV1ShROEvk5aptIT7bGmZunvc1OiKwJTXVbN0BiGm6k2zNW
}] }

Ethereum fallback mechanism in OP Stack

TheEthereum fallback mechanism allows rollups to "fall back" to Ethereum or another EVM chain in the case of downtime or errors submitting data to Celestia.

Implementation of fallback

The Ethereum fallback mechanism is implemented in thecelestiaorg/optimism v1.1.0 release.

Theop-batcher/batcher/driver.go andop-node/rollup/derive/calldata_source.go files are part of the Ethereum fallback mechanism in theop-batcher andop-node respectively.

Indriver.go , thecalldataTxCandidate function is responsible for the write pathof the Ethereum fallback. This function creates and submits a transaction to the batch inbox address with the given data. It uses the underlyingtxmgr to handle transaction sending and gas price management.

If the transaction data can be published as a blob to Celestia, it replaces the calldata with a blob identifier and sends the transaction with this data. If it cannot be published to Celestia, it falls back to Ethereum without any change to the transaction.

The blob identifier starts with the special prefix0xce , which was chosen a mnemonic for Celestia, and indicates that the remaining data has to interpreted as a little-endian encoded Block Height (8 bytes) and Blob Commitment (32 bytes). The combination of these can later be used to retrieve the original calldata from Celestia.

Prefix 8 bytes 32 bytes 0xce Block Height Blob Commitment go func (l * BatchSubmitter) sendTransaction (txdata txData, queue * txmgr.Queue[txData], receiptsCh chan txmgr.TxReceipt[txData],) { // ... } func (l * BatchSubmitter) sendTransaction (txdata txData, queue * txmgr.Queue[txData], receiptsCh chan txmgr.TxReceipt[txData],) { // ... } Incalldata_source.go ,theDataFromEVMTransactions function defines the read path of the Ethereum fallback. This function filters all of the transactions and returns the calldata from transactions that are sent to the batch inbox address from the batch sender address.

If the calldata matches the version prefix0xce , it is decoded as a blob identifier, the original calldata is retrieved from Celestia and returned for derivation. If the calldata does not match the prefix, the

entire calldata is returned for derivation.

go func

DataFromEVMTransactions (config * rollup.Config, batcherAddr common.Address, txs types.Transactions, log log.Logger) ([]eth.Data, error) { // ... } func

DataFromEVMTransactions (config * rollup.Config, batcherAddr common.Address, txs types.Transactions, log log.Logger) ([]eth.Data, error) { // ... } These two functions work together to ensure that the Ethereum fallback mechanism operates correctly, allowing the rollup to continue functioning even during periods of downtime on Celestia.

Testing the fallback

Testing out the Ethereum fallback mechanism can be done with thego-da tool. Triggering a simultaneous blob transaction will cause theop-batcher blob transaction to fail, with anincorrect account sequence error, which triggers a fallback to Ethereum.

To trigger the transaction, send this command from the samego/proto/da directory:

bash grpcurl

-proto

da.proto

-plaintext

-d

'{"blobs": [{"value": "SGVsbG8gd28ybGQh"}]}'

127.0 .0.1:26650

da.DAService.Submit grpcurl

-proto

da.proto

-plaintext

-d

'{"blobs": [{"value": "SGVsbG8gd28ybGQh"}]}'

127.0 .0.1:26650

da.DAService.Submit Alternatively, you can shut off thelocal-celestia-devnet and see that the OP Stack devnet logs show that the rollup has fallen back to the L1, in this case Ethereum, for posting data.

Span batches

Span batches can be enabled by settingOP_BATCHER_BATCH_TYPE: 1 in yourdocker-compose.yml file.

Note that this requires the Delta activation time to be configured. For your devnet, you should set"l2GenesisDeltaTimeOffset": "0x0", indevnetL1-template.json . This will enable span batches and can be tested by greppingdocker compose logs -f | grep batch_type which should includebatch_type=SpanBatch andbatch_type=1 . [][Edit this page on GitHub] Last updated: Previous page Deploy a dapp on Bubs testnet Next page Deploy an OP Stack devnet on Celestia []