How to Use the OP Stack with Avail

Introduction

Embark on setting up your own Avail-Optimism chain. This guide targets Ethereum's Sepolia testnet and Avail Turing testnet. For a detailed understanding, review theOptimism Documentation (opens in a new tab) .

Prerequisites

Ensure you have installed the following software.

Installation commands are based on Ubuntu 20.04 LTS: Software VersionGit (opens in a new tab) OS default Go (opens in a new tab) 1.21 Node (opens in a new tab) ^20 Pnpm (opens in a new tab) 8.5.6 Make (opens in a new tab) OS default jq (opens in a new tab) OS default direnv (opens in a new tab) Latest Foundry (opens in a new tab) ^0.2.0

Install Git

sudo

apt

install

-y

git

curl

make

jq

Install Go

wget

https://go.dev/dl/go1.20.linux-amd64.tar.gz tar

xvzf

go1.20.linux-amd64.tar.gz sudo

cp

go/bin/go

/usr/bin/go sudo

mv

go

/usr/lib echo

export

GOROOT=/usr/lib/go

~/.bashrc

Install Node.js

curl

-fsSL

https://deb.nodesource.com/setup_16.x

|

sudo

-E

bash

- sudo

apt-get

install

-y

nodejs

npm

Install Pnpm

sudo

npm

install

-g

pnpm

Install Make

sudo

apt

install

-y

make

Install jq

sudo

apt

install

-y

jq

Install direnv

sudo

apt

install

-y

direnv

Use the following releases while following the guide:

- op-node/v1.9.1
- op-proposer/v1.9.1
- op-batcher/v1.9.1
- op-geth v1.101408.0

Setup Avail DA-server

Run avail-da server (manually).

1. Navigate to

git clone https://github.com/availproject/avail-alt-da-server.git cd avail-alt-da-server 1. Build Avail DA Server:make da-server 2. Run your DA sidecar:./bin/avail-da-server ./cmd/avail --addr=localhost --port=8000 --avail.rpc= --avail.seed=<> --avail.appid=<>

Run using docker

- Copy.env.example
- to.env
- . Fill the values inside.
- Run the following commands:
- docker-compose build
- docker-compose up

Build the Adapter Source

1. Clone and navigate to the Avail adapter:
2. git
3. clone
4. https://github.com/ethereum-optimism/optimism.git
5. cd
6. optimism
7. git
8. fetch
9. --tag
10. --all
11. git
12. checkout
13. v1.9.1
14. git
15. submodule
16. update
17. --init
18. --recursive
19. Install modules:
20. pnpm
21. install
22. Compile the necessary packages:
23. make
24. op-node
25. op-batcher
26. op-proposer
27. pnpm
28. build

Build the Optimism Geth Source

1. Clone and navigate toop-geth
2. :
3. git
4. clone
5. https://github.com/ethereum-optimism/op-geth.git
6. cd

7. op-geth
8. git
9. fetch
10. --tag
11. --all
12. git
13. checkout
14. v1.101408.0
15. Compileop-geth
16. :
17. make
18. geth

Get Access to a Sepolia Node

For deploying to Sepolia, access an L1 node using a provider likeAlchemy (opens in a new tab) orrun your own Sepolia node (opens in a new tab) .

Generate and Secure Keys

Create four essential accounts with private keys:

- Admin
- (contract upgrade authority)
- Batcher
- (publishes Sequencer data to L1)
- Proposer
- (publishes L2 results to L1)
- Sequencer
- (signs blocks on the p2p network)

You can usecast wallet in thecontracts-bedrock package for key generation:

1. In the Optimism repo, navigate to thecontracts-bedrock package (opens in a new tab)
2. :
3. cd
4. ~/optimism/packages/contracts-bedrock
5. Generate accounts:
6. echo
7. "Admin:"
8. cast
9. wallet
10. new
11. echo
12. "Proposer:"
13. cast
14. wallet
15. new
16. echo
17. "Batcher:"
18. cast
19. wallet
20. new
21. echo
22. "Sequencer:"
23. cast
24. wallet
25. new
26. You should see an output similar to:
27. Admin:
28. Successfully
29. created
30. new
31. keypair.
32. Address:
33. 0xc4A01194958DE0D90A876e8A5fc9D7B530072148
34. Private
35. key:

36. 0xb8e39bd94a210e410c4024e1cc91014de45a5eb1e42f3aa99a368b5a5ac19b45
37. Proposer:
38. Successfully
39. created
40. new
41. keypair.
42. Address:
43. 0xFC0374Ae658e46cA4022acA179d3cb6D8e1A4934
44. Private
45. key:
46. 0xa9bc1b3f5deb1e00251df68bf86e3493b25bc5430665433546f2f9aacc748d1a
47. Batcher:
48. Successfully
49. created
50. new
51. keypair.
52. Address:
53. 0xD6857B5BE9468Be67d64ABaB48459378d5329b96
54. Private
55. key:
56. 0xe9cd8960fc7984a301d567b819e0c62871eb2c7239c2e66b8f319eaa45c3cbd5
57. Sequencer:
58. Successfully
59. created
60. new
61. keypair.
62. Address:
63. 0x33348817E4B1192D576C4f157e9a5EC93dc5392D
64. Private
65. key:
66. 0xd98b49e11e4e0be9931017831395e6644a50c36285d08e14d1a479af5ee08675
67. Record and securely store these key details. You'll need to fundAdmin
68. ,Proposer
69. , andBatcher
70. with Sepolia ETH (0.5 ETH forAdmin
71. , 0.2 ETH forProposer
72. , 0.1 ETH forBatcher
73.).

⚠ NOTE FOR PRODUCTION Use secure hardware for key management in production environments.cast wallet is not designed for production deployments.

Network Configuration and Setup

After building the repositories, configure your chain settings in thecontracts-bedrock package (opens in a new tab).

1. Ensure you are in thecontracts-bedrock
2. sub-directory:
3. cd
4. ~/optimism/packages/contracts-bedrock
5. Activate the environment withdirenv
6. :
7. If you need to installdirenv
8. , ensure you alsomodify the shell configuration (opens in a new tab)
9. .
10. direnv
11. allow
12. .

Core Contract Deployment

Deploy essential L1 contracts for the chain’s functionality:

1. Update/optimism/packages/contracts-bedrock/deploy-config
2. and update filegetting_started.json
3. .

cd packages/contracts-bedrock ./scripts/getting-started/config.sh 1. Add the following at the bottom of the config generated:

"useAltDA": true, "daCommitmentType": "GenericCommitment", "daChallengeWindow": 160, "daResolveWindow": 160, "daBondSize": 1000000, "daResolverRefundPercentage": 0 Example config (for reference purpose):

{ "l1StartingBlockTag": "0x2c9f156ae1cc024817b920b41c102b23b5a7526e16220c517078341f9890e8bd",

"l1ChainID": 11155111, "l2ChainID": 11155420, "l2BlockTime": 2, "l1BlockTime": 12,

"maxSequencerDrift": 600, "sequencerWindowSize": 3600, "channelTimeout": 300,

"p2pSequencerAddress": "0x15cb6e5fa6a7134F3f6f5a9130F70c1344162b77", "batchInboxAddress": "0xff0000000000000000000000000000000042069", "batchSenderAddress": "0x5bd31d51C4041108c238473f27BF5f72311b1bcF",

"l2OutputOracleSubmissionInterval": 120, "l2OutputOracleStartingBlockNumber": 0, "l2OutputOracleStartingTimestamp": 1725740628,

"l2OutputOracleProposer": "0x07B4c44290fD2Fba85d7924Fbf4A6499661bBe53", "l2OutputOracleChallenger": "0x79A953eAc59e40B3c7567a4cfc446c4B4912722f",

"finalizationPeriodSeconds": 12,

"proxyAdminOwner": "0x79A953eAc59e40B3c7567a4cfc446c4B4912722f", "baseFeeVaultRecipient": "0x79A953eAc59e40B3c7567a4cfc446c4B4912722f", "l1FeeVaultRecipient": "0x79A953eAc59e40B3c7567a4cfc446c4B4912722f", "sequencerFeeVaultRecipient": "0x79A953eAc59e40B3c7567a4cfc446c4B4912722f", "finalSystemOwner": "0x79A953eAc59e40B3c7567a4cfc446c4B4912722f", "superchainConfigGuardian": "0x79A953eAc59e40B3c7567a4cfc446c4B4912722f",

"baseFeeVaultMinimumWithdrawalAmount": "0x8ac7230489e80000", "l1FeeVaultMinimumWithdrawalAmount": "0x8ac7230489e80000", "sequencerFeeVaultMinimumWithdrawalAmount": "0x8ac7230489e80000", "baseFeeVaultWithdrawalNetwork": 0, "l1FeeVaultWithdrawalNetwork": 0, "sequencerFeeVaultWithdrawalNetwork": 0,

"gasPriceOracleOverhead": 0, "gasPriceOracleScalar": 1000000,

"enableGovernance": true, "governanceTokenSymbol": "OP", "governanceTokenName": "Optimism", "governanceTokenOwner": "0x79A953eAc59e40B3c7567a4cfc446c4B4912722f",

"l2GenesisBlockGasLimit": "0x1c9c380", "l2GenesisBlockBaseFeePerGas": "0x3b9aca00",

"eip1559Denominator": 50, "eip1559DenominatorCanyon": 250, "eip1559Elasticity": 6, "l2GenesisFjordTimeOffset": "0x0", "l2GenesisRegolithTimeOffset": "0x0", "l2GenesisEcotoneTimeOffset": "0x0", "l2GenesisDeltaTimeOffset": "0x0", "l2GenesisCanyonTimeOffset": "0x0", "systemConfigStartBlock": 0,

"requiredProtocolVersion": "0x0000000000000000000000000000000000000000000000000000000000000000", "recommendedProtocolVersion": "0x0000000000000000000000000000000000000000000000000000000000000000",

"faultGameAbsolutePrestate": "0x03c7ae758795765c6664a5d39bf63841c71ff191e9189522bad8ebff5d4eca98", "faultGameMaxDepth": 44, "faultGameClockExtension": 0, "faultGameMaxClockDuration": 1200, "faultGameGenesisBlock": 0, "faultGameGenesisOutputRoot": "0x0000000000000000000000000000000000000000000000000000000000000000", "faultGameSplitDepth": 14, "faultGameWithdrawalDelay": 600,

"preimageOracleMinProposalSize": 1800000, "preimageOracleChallengePeriod": 300,

"useAltDA": true, "daCommitmentType": "GenericCommitment", "daChallengeWindow": 160, "daResolveWindow": 160, "daBondSize": 1000000, "daResolverRefundPercentage": 0 } 1. Navigate to/optimism/packages/contracts-bedrock/deployments 2. , and createavail-optimism 3. directory:

cd

~/optimism/packages/contracts-bedrock/deployments mkdir

avail-optimism 1. Navigate to/optimism/packages/contracts-bedrock/ 2. and the deploy contracts (this can take up to 15 minutes):DEPLOYMENT_OUTFILE 3. = 4. deployments/artifact.json 5. \ 6. DEPLOY_CONFIG_PATH 7. = 8. deploy-config/getting-started.json 9. \ 10. forge 11. script 12. scripts/deploy/Deploy.s.sol:Deploy 13. --broadcast 14. --private-key 15. GS_ADMIN_PRIVATE_KEY \ 16. --rpc-url L1_RPC_URL 17. --slow 18. L2 Allocs

CONTRACT_ADDRESSES_PATH=deployments/artifact.json DEPLOY_CONFIG_PATH=deploy-config/getting-started.json STATE_DUMP_PATH=deploy-config/statedump.json forge script scripts/L2Genesis.s.sol:L2Genesis --sig 'runWithStateDump()' --chain

⚠ If you see a nondescript error that includesEvmError: Revert and Script failed then you likely need to change theIMPL_SALT environment variable. This variable determines the addresses of various smart contracts that are deployed

viaCREATE2 .

If the sameIMPL_SALT is used to deploy the same contracts twice, the second deployment will fail.

You can generate a newIMPL_SALT by runningdirenv reload anywhere in the Avail Optimism Monorepo.

Setting Up L2 Configuration

After configuring the L1 layer, focus shifts to establishing the L2 infrastructure. This involves generating three key files:

- genesis.json
- for the genesis block
- rollup.json
- for rollup configurations
- jwt.txt
- for secure communication betweenop-node

- andop-geth

- Navigate to theop-node

- directory:
- cd
- ~/optimism/op-node
- Run the following command, ensuring you replace
- with your specific L1 RPC URL. This generates thegenesis.json
- androllup.json
- files:
- go
- run
- cmd/main.go
- genesis
- l2
- \
- --deploy-config
- ../packages/contracts-bedrock/deploy-config/getting-started.json
- \
- --l1-deployments
- ../packages/contracts-bedrock/deployments/artifact.json
- \
- --outfile.l2
- genesis.json
- \
- --outfile.rollup
- rollup.json
- \
- --l1-rpc L1_RPC_URL \
- --l2-allocs
- ../packages/contracts-bedrock/deploy-config/statedump.json
- You'll find the newly createdgenesis.json
- androllup.json
- in theop-node
- package.
- Add the following at the end ofrollup.json
- :

"alt_da": { "da_challenge_contract_address": "0x0000000000000000000000000000000000000000", "da_commitment_type": "GenericCommitment", "da_challenge_window": 160, "da_resolve_window": 160 } 1. Generate ajwt.txt 2. file, which is crucial for the secure interaction between nodes: 3. openssl 4. rand 5. -hex 6. 32 7. > 8. jwt.txt 9. To getop-geth 10. ready, move thegenesis.json 11. andjwt.txt 12. files into its directory: 13. cp 14. genesis.json 15. ~/op-geth 16. cp 17. jwt.txt 18. ~/op-geth

These steps ensure the L2 layer is correctly configured and ready for integration with the L1 components, paving the way for a fully functional EVM Rollup on the Avail-OP Stack.

Initialize and Configure Geth

Prepareop-geth for running the chain:

1. Navigate toop-geth
2. :
3. cd
4. ~/op-geth
5. Create a data directory:
6. mkdir
7. datadir
8. Initialize with the genesis file:
9. build/bin/geth
10. init
11. --datadir=datadir
12. genesis.json

Running op-geth

To initiateop-geth , navigate to its directory and execute the following commands:

cd

~/op-geth ./build/bin/geth \ --datadir

./datadir \ --http \ --http.corsdomain= "" \ *--http.vhosts= "* \ --http.addr=0.0.0.0 \ --http.port=9545 \ --http.api=web3,debug,eth,txpool,net,engine \ --ws \ --ws.addr=0.0.0.0 \ --ws.port=9546 \ --ws.origins= "" \ *--ws.api=debug,eth,txpool,net,engine* \ *--syncmode=full* \ *--nodiscover* \ *--maxpeers=0* \ *--networkid=42069* \ *--authrpc.vhosts= "*" \ --authrpc.addr=0.0.0.0 \ --authrpc.port=9551 \ --authrpc.jwtsecret=./jwt.txt \ --rollup.disabletxpoolgossip=true \ --state.scheme=hash op-geth is now active, but block creation will begin onceop-node is operational.

Why Archive Mode?

Archive mode, requiring more disk space than full mode, is essential for:

1. op-proposer
2. to access the full state history.
3. The explorer's functionality.

Reinitializing op-geth

In cases of database corruption indicated byop-node errors or failure to find L2 heads, follow these steps:

1. Stopop-geth
2. .
3. Remove the existing data:cd
4. ~/op-geth
5. rm
6. -rf
7. datadir/geth
8. Reinitialize:build/bin/geth
9. init
10. --datadir=datadir
11. genesis.json
12. Restartop-geth
13. and thenop-node
14. .

Running op-node

To launchop-node , which acts as a consensus client, run:

cd

~/optimism/op-node ./bin/op-node \ --l2=http://localhost:9551 \ --l2.jwt-secret=./jwt.txt \ --sequencer.enabled \ --sequencer.l1-confs=5 \ --verifier.l1-confs=4 \ --rollup.config=./rollup.json \ --rpc.addr=0.0.0.0 \ --rpc.port=8547 \ --p2p.disable \ --rpc.enable-admin \ --p2p.sequencer.key=GS_SEQUENCER_PRIVATE_KEY \ --l1=L1_RPC_URL \ --l1.rpckind=L1_RPC_KIND \ --altda.enabled=true \ --altda.da-server= < DA_SERVER_HTTP_URL

\ --altda.da-service=true \ --l1.beacon.ignore=true Block creation will commence onceop-node starts processing L1 information and interfaces withop-geth .

P2P Synchronization

To optimize synchronization and avoid network resource waste:

- Disable p2p sync (--p2p.disable
-) by default.
- Use specific command line parameters for synchronization among multiple nodes.

Running op-batcher

op-batcher is crucial in publishing transactions from the Sequencer to L1. Ensure it has at least 1 Sepolia ETH for operational continuity.

cd

~/optimism/op-batcher ./bin/op-batcher \ --l2-eth-rpc=http://localhost:9545 \ --rollup-rpc=http://localhost:8547 \ --poll-interval=1s \ --sub-safety-margin=6 \ --num-confirmations=1 \ --safe-abort-nonce-too-low-count=3 \ --resubmission-timeout=30s \ --rpc.addr=0.0.0.0 \ --rpc.port=8548 \ --rpc.enable-admin \ --max-channel-duration=1 \ --l1-eth-rpc=L1_RPC_URL \ --private-key=GS_BATCHER_PRIVATE_KEY \ --altda.enabled=true \ --altda.da-service=true \ --altda.da-server= < DA_SERVER_HTTP_URL

Controlling Batcher Costs

Adjust the--max-channel-duration=n setting to balance transaction frequency on L1 and the operational costs of the batcher. Recommended is a minumum of 2 since avail block time is 20s and ethereum's 12sec.

Running op-proposer

Finally, startop-proposer to propose new state roots:

cd

~/optimism/op-proposer ./bin/op-proposer \ --poll-interval=12s \ --rpc.port=9560 \ --rollup-rpc=http://localhost:8547 \ --l2oo-address=L2OO_ADDR \ --private-key=PROPOSER_KEY \ --l1-eth-rpc=L1_RPC

Acquire Sepolia ETH for Layer 2

To obtain ETH on your Rollup:

1. Go tocontracts-bedrock
2. :
3. cd
4. ~/optimism/packages/contracts-bedrock
5. Find the L1 standard bridge contract address:
6. cat
7. deployments/avail-optimism/L1StandardBridgeProxy.json
8. |
9. jq
10. -r
11. .address
12. Send Sepolia ETH to the bridge contract address.

Conduct Test Transactions

You now have a fully operational Avail-Powered Optimism-based EVM Rollup. Experiment with it as you would with any other test blockchain.

Congratulations on setting up your chain!

Overview OP Stack