Creating Your Own L2 Rollup Testnet

Pleasebe prepared to set aside approximately one hour to get everything running properly andmake sure to read through the guide carefully . You don't want to miss any important steps that might cause issues down the line. This tutorial isdesigned for developers who want to learn about the OP Stack by spinning up an OP Stack testnet chain. You'll walk through the full deployment process and teach you all of the components that make up the OP Stack, andyou'll end up with your very own OP Stack testnet .

You can use this testnet to experiment and perform tests, or you can choose to modify the chain to adapt it to your own needs.The OP Stack is free and open source software licensed entirely under the MIT license . You don't need permission from anyone to modify or deploy the stack in any configuration you want.

⚠ Modifications to the OP Stack may prevent a chain from being able to benefit from aspects of theOptimism Superchain . Make sure to check out theSuperchain Explainer to learn more.

What You're Going to Deploy

When deploying an OP Stack chain, you'll be setting up four different components. It's useful to understand what each of these components does before you start deploying your chain.

Smart Contracts

The OP Stack gives you the ability to deploy your own Rollup chains that use a Layer 1 blockchain to host and order transaction data. OP Stack chains use several smart contracts on the L1 blockchain to manage aspects of the Rollup. Each OP Stack chain has its own set of L1 smart contracts that are deployed when the chain is created. You'll be using the L1 smart contracts found in thecontracts-bedrock package(opens in a new tab) within theOptimism Monorepo(opens in a new tab) .

Sequencer Node

OP Stack chains use Sequencer nodes to gather proposed transactions from users and publish them to the L1 blockchain. Vanilla (unmodified) OP Stack chains rely on at least one of these Sequencer nodes, so you'll have to run one. You can also run additional non-Sequencer nodes if you'd like (not included in this tutorial).

Consensus Client

OP Stack nodes, like Ethereum nodes, have a consensus client. The consensus client is responsible for determining the list and ordering of blocks and transactions that are part of your blockchain. Several implementations of the OP Stack consensus client exist, includingop-node (maintained by Optimism Foundation),magi (opens in a new tab) (maintained by a16z) andhildr (opens in a new tab) (maintained by OptimismJ). In this tutorial you'll be using theop-node implementation(opens in a new tab) found within theOptimism Monorepo(opens in a new tab) .

Execution Client

OP Stack nodes, like Ethereum nodes, also have an execution client. The execution client is responsible for executing transactions and storing/updating the state of the blockchain. Various implementations of the OP Stack execution client exist, includingop-geth (maintained by Optimism Foundation),op-erigon (opens in a new tab) (maintained by Test in Prod), andop-nethermind (coming soon). In this tutorial you'll be using theop-geth implementation(opens in a new tab) found within theop-geth repository(opens in a new tab) .

Batcher

The Batcher is a service that publishes transactions from the Sequencer to the L1 blockchain. The Batcher runs continuously alongside the Sequencer and publishes transactions in batches (hence the name) on a regular basis. You'll be using theop-batcher implementation(opens in a new tab) of the Batcher component found within theOptimism Monorepo(opens in a new tab) .

Proposer

The Proposer is a service responsible for publishing transactionsresults (in the form of L2 state roots) to the L1 blockchain. This allows smart contracts on L1 to read the state of the L2, which is necessary for cross-chain communication and reconciliation between state changes. It's likely that the Proposer will be removed in the future, but for now it's a necessary component of the OP Stack. You'll be using theop-proposer implementation(opens in a new tab) of the Proposer component found within theOptimism Monorepo(opens in a new tab) .

Software Dependencies

Dependency Version Version Check Commandgit(opens in a new tab) ^2 git --versiongo(opens in a new tab) ^1.21 go versionnode(opens in a new tab) ^20 node --versionpnpm(opens in a new tab) ^8 pnpm --versionfoundry(opens in a new tab) ^0.2.0 forge --versionmake(opens in a new tab) ^3 make --versionjq(opens in a new tab) ^1.6 jq --versiondirenv(opens in a new tab) ^2 direnv --version

Notes on Specific Dependencies

node

We recommend using the latest LTS version of Node.js (currently v20)nvm (opens in a new tab) is a useful tool that can help you manage multiple versions of Node.js on your machine. You may experience unexpected errors on older versions of Node.js.

direnv

Parts of this tutorial usedirenv (opens in a new tab) as a way of loading environment variables from.envrc files into your shell. This means you won't have to manually export environment variables every time you want to use them.direnv only ever has access to files that you explicitly allow it to see.

Afterinstallingdirenv (opens in a new tab) , you will need tomake sure thatdirenv is hooked into your shell(opens in a new tab) . Make sure you've followedthe guide on thedirenv website(opens in a new tab) , thenclose your terminal and reopen it so that the changes take effect (orsource your config file if you know how to do that).

Make sure that you have correctly hookeddirenv into your shell by modifying your shell configuration file (like~/.bashrc or~/.zshrc). If you haven't edited a config file then you probably haven't configureddirenv properly (and things might not work later).

Get Access to a Sepolia Node

You'll be deploying a OP Stack Rollup chain that uses a Layer 1 blockchain to host and order transaction data. The OP Stack Rollups were designed to use EVM Equivalent blockchains like Ethereum, OP Mainnet, or standard Ethereum testnets as their L1 chains.

This guide uses the Sepolia testnet as an L1 chain . We recommend that you also use Sepolia. You can also use other EVM-compatible blockchains, but you may run into unexpected errors. If you want to use an alternative network, make sure to carefully review each command and replace any Sepolia-specific values with the values for your network.

Since you're deploying your OP Stack chain to Sepolia, you'll need to have access to a Sepolia node. You can either use a node provider likeAlchemy(opens in a new tab) (easier) or run your own Sepolia node (harder).

Build the Source Code

You're going to be spinning up your OP Stack chain directly from source code instead of using a container system likeDocker(opens in a new tab) . Although this adds a few extra steps, it means you'll have an easier time modifying the behavior of the stack if you'd like to do so. If you want a summary of the various components you'll be using, take another look at theWhat You're Going to Deploy section above.

You're using the home directory~/ as the work directory for this tutorial for simplicity. You can use any directory you'd like but using the home directory will allow you to copy/paste the commands in this guide. If you choose to use a different directory, make sure you're using the correct directory in the commands throughout this tutorial.

Build the Optimism Monorepo

Clone the Optimism Monorepo

cd

~ git

clone

https://github.com/ethereum-optimism/optimism.git

Enter the Optimism Monorepo

cd

optimism

Check out the correct branch

You will be using thetutorials/chain branch of the Optimism Monorepo to deploy an OP Stack testnet chain during this tutorial. This is a non-production branch that lags behind thedevelop branch. You shouldNEVER use thedevelop ortutorials/chain branches in production. git

checkout

tutorials/chain

Check your dependencies

Don't skip this step! Make sure you have all of the required dependencies installed before continuing. Run the following script and double check that you have all of the required versions installed. If you don't have the correct versions installed, you may run into unexpected errors.

./packages/contracts-bedrock/scripts/getting-started/versions.sh

Install dependencies

pnpm

install

Build the various packages inside of the Optimism Monorepo

make

op-node

op-batcher

op-proposer pnpm

build

Buildop-geth

Clone op-geth

cd

~ git

clone

https://github.com/ethereum-optimism/op-geth.git

Enter op-geth

cd

op-geth

Build op-geth

make

geth

Fill Out Environment Variables

You'll need to fill out a few environment variables before you can start deploying your chain.

Enter the Optimism Monorepo

cd

~/optimism

Duplicate the sample environment variable file

cp

.envrc.example

.envrc

Fill out the environment variable file

Open up the environment variable file and fill out the following variables:

Variable Name Description L1_RPC_URL URL for your L1 node (a Sepolia node in this case). L1_RPC_KIND Kind of L1 RPC you're connecting to, used to inform optimal transactions receipts fetching. Valid options:alchemy ,quicknode ,infura ,parity ,nethermind ,debug_geth ,erigon ,basic ,any .

Generate Addresses

You'll need four addresses and their private keys when setting up the chain:

- TheAdmin
- address has the ability to upgrade contracts.
- TheBatcher
- address publishes Sequencer transaction data to L1.
- TheProposer
- address publishes L2 transaction results (state roots) to L1.
- TheSequencer
- address signs blocks on the p2p network.

Enter the Optimism Monorepo

cd

~/optimism

Generate new addresses

You shouldnot use thewallets.sh tool for production deployments. If you are deploying an OP Stack based chain into production, you should likely be using a combination of hardware security modules and hardware wallets. ./packages/contracts-bedrock/scripts/getting-started/wallets.sh

Check the output

Make sure that you see output that looks something like the following:

Copy the following into your .envrc file:

Admin address

export GS_ADMIN_ADDRESS=0x9625B9aF7C42b4Ab7f2C437dbc4ee749d52E19FC export GS_ADMIN_PRIVATE_KEY=0xbb93a75f64c57c6f464fd259ea37c2d4694110df57b2e293db8226a502b30a34

Batcher address

export GS_BATCHER_ADDRESS=0xa1AEF4C07AB21E39c37F05466b872094edcf9cB1 export GS_BATCHER_PRIVATE_KEY=0xe4d9cd91a3e53853b7ea0dad275efdb5173666720b1100866fb2d89757ca9c5a

Proposer address

export GS_PROPOSER_ADDRESS=0x40E805e252D0Ee3D587b68736544dEfB419F351b export GS_PROPOSER_PRIVATE_KEY=0x2d1f265683ebe37d960c67df03a378f79a7859038c6d634a61e40776d561f8a2

Sequencer address

export GS_SEQUENCER_ADDRESS=0xC06566E8Ec6cF81B4B26376880dB620d83d50Dfb export GS_SEQUENCER_PRIVATE_KEY=0x2a0290473f3838dbd083a5e17783e3cc33c905539c0121f9c76614dda8a38dca

Save the addresses

Copy the output from the previous step and paste it into your.envrc file as directed.

Fund the addresses

You will need to send ETH to theAdmin ,Proposer , andBatcher addresses. The exact amount of ETH required depends on the L1 network being used.You do not need to send any ETH to theSequencer address as it does not send transactions.

It's recommended to fund the addresses with the following amounts when using Sepolia:

- Admin
- — 0.5 Sepolia ETH
- Proposer
- — 0.2 Sepolia ETH
- Batcher
- — 0.1 Sepolia ETH

Load Environment variables

Now that you've filled out the environment variable file, you need to load those variables into your terminal.

Enter the Optimism Monorepo

cd

~/optimism

Load the variables with direnv

You're about to usedirenv to load environment variables from the.envrc file into your terminal. Make sure that you've installeddirenv (opens in a new tab) and that you've properlyhookeddirenv into your shell . Next you'll need to allowdirenv to read this file and load the variables into your terminal using the following command.

direnv

allow WARNING:direnv will unload itself whenever your.envrc file changes.Youmust rerun the following command every time you change the.envrc file.

Confirm that the variables were loaded

After runningdirenv allow you should see output that looks something like the following (the exact output will vary depending on the variables you've set, don't worry if it doesn't look exactly like this):

direnv:

loading

~/optimism/.envrc direnv:

export

+DEPLOYMENT_CONTEXT

+ETHERSCAN_API_KEY

+GS_ADMIN_ADDRESS

+GS_ADMIN_PRIVATE_KEY

+GS_BATCHER_ADDRESS

+GS_BATCHER_PRIVATE_KEY

+GS_PROPOSER_ADDRESS

+GS_PROPOSER_PRIVATE_KEY

+GS_SEQUENCER_ADDRESS

+GS_SEQUENCER_PRIVATE_KEY

+IMPL_SALT

+L1_RPC_KIND

+L1_RPC_URL

+PRIVATE_KEY

+TENDERLY_PROJECT

+TENDERLY_USERNAME If you don't see this output, you likely haven't properly configureddirenv . Make sure you've configureddirenv properly and rundirenv allow again so that you see the desired output.

Configure your network

Once you've built both repositories, you'll need to head back to the Optimism Monorepo to set up the configuration file for your chain. Currently, chain configuration lives inside of the contracts-bedrock (opens in a new tab) package in the form of a JSON file.

Enter the Optimism Monorepo

cd

~/optimism

Move into the contracts-bedrock package

cd

packages/contracts-bedrock

Generate the configuration file

Run the following script to generate thegetting-started.json configuration file inside of thedeploy-config directory.

./scripts/getting-started/config.sh

Review the configuration file (Optional)

If you'd like, you can review the configuration file that was just generated by opening updeploy-config/getting-started.json in your favorite text editor. It's recommended to keep this file as-is for now so you don't run into any unexpected errors.

Deploy the Create2 Factory (Optional)

If you're deploying an OP Stack chain to a network other than Sepolia, you may need to deploy a Create2 factory contract to the L1 chain. This factory contract is used to deploy OP Stack smart contracts in a deterministic fashion.

This step is typically only necessary if you are deploying your OP Stack chain to custom L1 chain. If you are deploying your OP Stack chain to Sepolia, you can safely skip this step.

Check if the factory exists

The Create2 factory contract will be deployed at the address0x4e59b44847b379578588920cA78FbF26c0B4956C . You can check if this contract has already been deployed to your L1 network with a block explorer or by running the following command:

cast

codesize

0x4e59b44847b379578588920cA78FbF26c0B4956C

--rpc-url L1_RPC_URL If the command returns0 then the contract has not been deployed yet. If the command returns69 then the contract has been deployed and you can safely skip this section.

Fund the factory deployer

You will need to send some ETH to the address that will be used to deploy the factory contract,0x3fAB184622Dc19b6109349B94811493BF2a45362 . This address can only be used to deploy the factory contract and will not be used for anything else. Send at least 1 ETH to this address on your L1 chain.

Deploy the factory

Usingcast , deploy the factory contract to your L1 chain:

cast

publish

--rpc-url L1_RPC_URL
0xf8a58085174876e800830186a08080b853604580600e600039806000f350fe7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffe03601600081602082378035828234f58015156039578182fd5b808252505050601

Wait for the transaction to be mined

Make sure that the transaction is included in a block on your L1 chain before continuing.

Verify that the factory was deployed

Run the code size check again to make sure that the factory was properly deployed:

cast

codesize

0x4e59b44847b379578588920cA78FbF26c0B4956C

--rpc-url L1_RPC_URL

Deploy the L1 contracts

Once you've configured your network, it's time to deploy the L1 contracts necessary for the functionality of the chain.

Deploy the L1 contracts

forge

script

scripts/Deploy.s.sol:Deploy

--private-key GS_ADMIN_PRIVATE_KEY --broadcast

--rpc-url L1_RPC_URL --slow If you see a nondescript error that includesEvmError: Revert andScript failed then you likely need to change theIMPL_SALT environment variable. This variable determines the addresses of various smart contracts that are deployed viaCREATE2(opens in a new tab) . 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 allow anywhere in the Optimism Monorepo.

Generate contract artifacts

forge

script

scripts/Deploy.s.sol:Deploy

--sig

'sync()'

--rpc-url L1_RPC_URL

Generate the L2 config files

Now that you've set up the L1 smart contracts you can automatically generate several configuration files that are used within the Consensus Client and the Execution Client.

You need to generate three important files:

1. genesis.json
2. includes the genesis state of the chain for the Execution Client.
3. rollup.json
4. includes configuration information for the Consensus Client.
5. jwt.txt
6. is aJSON Web Token(opens in a new tab)
7. that allows the Consensus Client and the Execution Client to communicate securely (the same mechanism is used in Ethereum clients).

Navigate to the op-node package

cd

~/optimism/op-node

Create genesis files

Now you'll generate thegenesis.json androllup.json files within theop-node folder:

go

run

cmd/main.go

genesis

l2 \ --deploy-config

../packages/contracts-bedrock/deploy-config/getting-started.json \ --deployment-dir

../packages/contracts-bedrock/deployments/getting-started/ \ --outfile.l2

genesis.json \ --outfile.rollup

rollup.json \ --l1-rpc L1_RPC_URL

Create an authentication key

Next you'll create aJSON Web Token(opens in a new tab) that will be used to authenticate the Consensus Client and the Execution Client. This token is used to ensure that only the Consensus Client and the Execution Client can communicate with each other. You can generate a JWT with the following command:

openssl

rand

-hex

32

jwt.txt

Copy genesis files into the op-geth directory

Finally, you'll need to copy thegenesis.json file andjwt.txt file intoop-geth so you can use it to initialize and runop-geth :

cp

genesis.json

~/op-geth cp

jwt.txt

~/op-geth

Initializeop-geth

You're almost ready to run your chain! Now you just need to run a few commands to initializeop-geth . You're going to be running a Sequencer node, so you'll need to import theSequencer private key that you generated earlier. This private key is what your Sequencer will use to sign new blocks.

Navigate to the op-geth directory

cd

~/op-geth

Create a data directory folder

mkdir

datadir

Initialize op-geth

build/bin/geth

init

--datadir=datadir

genesis.json

Startop-geth

Now you'll startop-geth , your Execution Client. Note that you won't start seeing any transactions until you start the Consensus Client in the next step.

Open up a new terminal

You'll need a terminal window to runop-geth in.

Navigate to the op-geth directory

cd

~/op-geth

Run op-geth

You're using--gcmode=archive to runop-geth here because this node will act as your Sequencer. It's useful to run the Sequencer in archive mode because theop-proposer requires access to the full state. Feel free to run other (non-Sequencer) nodes in full mode if you'd like to save disk space. It's important that you've already initialized the geth node at this point as per the previous section. Failure to do this will cause startup issues betweenop-geth andop-node . ./build/bin/geth \ --datadir

./datadir \ --http \ --http.corsdomain= "" | --*http.vhosts= "* \ --http.addr=0.0.0.0 \ --http.api=web3,debug,eth,txpool,net,engine \ --ws \ --ws.addr=0.0.0.0 \ --ws.port=8546 \ --ws.origins="*"*| --*ws.api=debug,eth,txpool,net,engine* \ --*syncmode=full* \ --*gcmode=archive* \ --*nodiscover* \ --*maxpeers=0* \ --*networkid=42069* \ --*authrpc.vhosts= "* \ --authrpc.addr=0.0.0.0 \ --authrpc.port=8551 \ --authrpc.jwtsecret=./jwt.txt \ --rollup.disabletxpoolgossip=true

Startop-node

Once you've gotop-geth running you'll need to runop-node . Like Ethereum, the OP Stack has a Consensus Client (op-node) and an Execution Client (op-geth). The Consensus Client "drives" the Execution Client over the Engine API.

Open up a new terminal

You'll need a terminal window to run theop-node in.

Navigate to the op-node directory

cd

~/optimism/op-node

Run op-node

./bin/op-node \ --l2=http://localhost:8551 \ --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 Once you run this command, you should start seeing theop-node begin to sync L2 blocks from the L1 chain. Once theop-node has caught up to the tip of the L1 chain, it'll begin to send blocks toop-geth for execution. At that point, you'll start to see blocks being created inside ofop-geth .

By default, yourop-node will try to use a peer-to-peer to speed up the synchronization process. If you're using a chain ID that is also being used by others, like the default chain ID for this tutorial (42069), yourop-node will receive blocks signed by other sequencers. These requests will fail and waste time and network resources.To avoid this, this tutorial starts with peer-to-peer synchronization disabled (--p2p.disable).

Once you have multiple nodes, you may want to enable peer-to-peer synchronization. You can add the following options to theop-node command to enable peer-to-peer synchronization with specific nodes:

--p2p.static= \ --p2p.listen.ip=0.0.0.0 \ --p2p.listen.tcp=9003 \ --p2p.listen.udp=9003 \ You can alternatively also remove the--p2p.static option but you may see failed requests from other chains using the same chain ID.

Startop-batcher

Theop-batcher takes transactions from the Sequencer and publishes those transactions to L1. Once these Sequencer transactions are included in a finalized L1 block, they're officially part of the canonical chain. Theop-batcher is critical!

It's best to give theBatcher address at least 1 Sepolia ETH to ensure that it can continue operating without running out of ETH for gas. Keep an eye on the balance of theBatcher address because it can expend ETH quickly if there are a lot of transactions to publish.

Open up a new terminal

You'll need a terminal window to run theop-batcher in.

Navigate to the op-batcher directory

cd

~/optimism/op-batcher

Run op-batcher

./bin/op-batcher \ --l2-eth-rpc=http://localhost:8545 \ --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 The--max-channel-duration=n setting tells the batcher to write all the data to L1 everyn L1 blocks. When it is low, transactions are written to L1 frequently and other nodes can synchronize from L1 quickly. When it is high, transactions are written to L1 less frequently and the batcher spends less ETH. If you want to reduce costs, either set this value to 0 to disable it or increase it to a higher value.

Startop-proposer

Now startop-proposer , which proposes new state roots.

Open up a new terminal

You'll need a terminal window to run theop-proposer in.

Navigate to the op-proposer directory

cd

~/optimism/op-proposer

Run op-proposer

./bin/op-proposer \ --poll-interval=12s \ --rpc.port=8560 \ --rollup-rpc=http://localhost:8547 \ --l2oo-address= (cat

../packages/contracts-bedrock/deployments/getting-started/L2OutputOracleProxy.json

|

jq

-r

.address) \ --private-key=GS_PROPOSER_PRIVATE_KEY \ --l1-eth-rpc=L1_RPC_URL

Connect Your Wallet to Your Chain

You now have a fully functioning OP Stack Rollup with a Sequencer node running onhttp://localhost:8545 . You can connect your wallet to this chain the same way you'd connect your wallet to any other EVM chain. If you need an easy way to connect to your chain, justclick here(opens in a new tab) .

Get ETH On Your Chain

Once you've connected your wallet, you'll probably notice that you don't have any ETH to pay for gas on your chain. The easiest way to deposit Sepolia ETH into your chain is to send ETH directly to theL1StandardBridge contract.

Navigate to the contracts-bedrock directory

cd

~/optimism/packages/contracts-bedrock

Get the address of the L1StandardBridgeProxy contract

cat

deployments/getting-started/L1StandardBridgeProxy.json

|

jq

-r

.address

Send some Sepolia ETH to the L1StandardBridgeProxy contract

Grab the L1 bridge proxy contract address and, using the wallet that you want to have ETH on your Rollup, send that address a small amount of ETH on Sepolia (0.1 or less is fine). This will trigger a deposit that will mint ETH into your wallet on L2. It may take up to 5 minutes for that ETH to appear in your wallet on L2.

See Your Rollup in Action

You can interact with your Rollup the same way you'd interact with any other EVM chain. Send some transactions, deploy some contracts, and see what happens!

Next Steps

- You canmodify the blockchain in various ways
- .
- Check out theprotocol specs(opens in a new tab)
- for more detail about the rollup protocol.
- If you run into any problems, please visit theChain Operators Troubleshooting Guide
- for help.

Overview Using the Optimism SDK