

# How to Use the OP Stack with Avail

## Introduction

Embark on setting up your own Avail-Optimism chain. This guide targets Ethereum's Goerli testnet and Avail Goldberg testnet. For a detailed understanding, review the [Optimism Documentation \(opens in a new tab\)](#).

In this guide, you will conduct the following:

- [Introduction](#)
- [Prerequisites](#)
- [Compile the Core Codebase](#)
- - [Build the Adapter Source](#)
- - [Build the Optimism Geth Source](#)
- - [Get Access to a Goerli Node](#)
- [Generate and Secure Keys](#)
- [Network Configuration and Setup](#)
- [Core Contract Deployment](#)
- [Setting Up L2 Configuration](#)
- [Initialize and Configure Geth](#)
- [Launch and Monitor Nodes](#)
- - [Running op-geth](#)
- - - [Why Archive Mode?](#)
- - - [Reinitializing op-geth](#)
- - [Running op-node](#)
- - - [P2P Synchronization](#)
- - [Running op-batcher](#)
- - - [Controlling Batchers Costs](#)
- - [Running op-proposer](#)
- [Acquire Goerli ETH for Layer 2](#)
- [Conduct Test Transactions](#)

## Prerequisites

Ensure you have installed the following software.

Installation commands are based on Ubuntu 20.04 LTS: Software Version [Git \(opens in a new tab\)](#) OS default [Go \(opens in a new tab\)](#) 1.20 [Node \(opens in a new tab\)](#) 16.19.0 [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 \(opens in a new tab\)](#) Foundry will be installed locally within the project's environment, viapackage.json

## 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](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

## Compile the Core Codebase

Setting up the EVM Rollup requires compiling code from two critical repositories: the [avail-op-stack-adapter monorepo \(opens in a new tab\)](#) and the [op-geth repository \(opens in a new tab\)](#).

### Build the Adapter Source

1. Clone and navigate to the Avail adapter:
2. git
3. clone
4. <https://github.com/availproject/avail-op-stack-adapter.git>
5. cd
6. avail-op-stack-adapter
7. Install modules:
8. pnpm
9. install
10. Compile the necessary packages:
11. make
12. op-node
13. op-batcher
14. op-proposer
15. pnpm
16. 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. Compileop-geth
9. :
10. make
11. geth

## Get Access to a Goerli Node

For deploying to Goerli, access an L1 node using a provider like [Alchemy \(opens in a new tab\)](#) or [run your own Goerli 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 Avail adapter repo, navigate to the[contracts-bedrock package \(opens in a new tab\)](#)
2. :
3. cd
4. ~/avail-op-stack-adapter/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. Batchers:
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. , andBatchers
70. with Goerli ETH (2 ETH forAdmin
71. , 5 ETH forProposer
72. , 10 ETH forBatchers
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 the [contracts-bedrock package \(opens in a new tab\)](#).

1. Ensure you are in thecontracts-bedrock
2. sub-directory:
3. cd
4. ~/avail-op-stack-adapter/packages/contracts-bedrock
5. Copy the environment file:
6. cp
7. .envrc.example
8. .envrc
9. Edit.envrc
10. with necessary values:
11.
  - ETH\_RPC\_URL
12.
  - : URL for your L1 node.
13.
  - PRIVATE\_KEY
14.
  - : Private key of the Admin account.
15.
  - DEPLOYMENT\_CONTEXT
- 16.

- : Name of the network; should be "avail-optimism"

17. Activate the environment with `direnv`
18. :
19. If you need to install `direnv`
20. , ensure you also [modify the shell configuration \(opens in a new tab\)](#)
21. .
22. `direnv`
23. `allow`
24. .
25. Choose an L1 block as a starting point using `cast`
26. command:
27. Using a finalized L1 block as our starting block is best.
28. `cast`
29. `block`
30. `finalized`
31. `--rpc-url`
32. `ETH_RPC_URL`
33. |
34. `grep`
35. `-E`
36. `"(timestamp|hash|number)"`
37. You should get a response that looks like:
38. `hash`
39. `0x784d8e7f0e90969e375c7d12dac7a3df6879450d41b4cb04d4f8f209ff0c4cd9`
40. `number`
41. `8482289`
42. `timestamp`
43. `1676253324`
44. Next, create and modify `deploy-config/avail-optimism.json`
45. based on `deploy-config/getting-started.json`
46. . Retain the default settings provided in the configuration and apply the following modifications:
47.
  - Replace `ADMIN`
48.
  - with the address of the Admin account you generated earlier.
49.
  - Replace `PROPOSER`
50.
  - with the address of the Proposer account you generated earlier.
51.
  - Replace `BATCHER`
52.
  - with the address of the Batchers account you generated earlier.
53.
  - Replace `SEQUENCER`
54.
  - with the address of the Sequencer account you generated earlier.
55.
  - Replace `BLOCKHASH`
56.
  - with the blockhash you got from the `cast` command.
57.
  - Replace `TIMESTAMP`
58.
  - with the timestamp you got from the `cast` command. Note that although all the other fields are strings, this field is a number! Don't include the quotation marks.
59. Configure `enableDA`
60. in `avail-optimism.json`
61. `(true`
62. for Avail chain as `DA,false`
63. for Ethereum).
64. Enter `op-avail`
65. module:
66. `cd`
67. `~/avail-op-stack-adapter/op-avail`
68. Create `config.json`
69. with necessary variables (seed
70. , `api_url`

```

71. ,app_id
72. ).
73. {
74. "seed"
75. :
76. "test test test test test test test test test test test test"
77. ,
78. "api_url"
79. :
80. "wss://goldberg.avail.tools:443/ws"
81. ,
82. "app_id"
83. :
84. 1
85. }

```

## Core Contract Deployment

Deploy essential L1 contracts for the chain's functionality:

1. Navigate to/avail-op-stack-adapter/packages/contracts-bedrock/deployments
2. , and createavail-optimism
3. directory:

cd

~/avail-op-stack-adapter/packages/contracts-bedrock/deployments mkdir

avail-optimism 1. Navigate to/avail-op-stack-adapter/packages/contracts-bedrock/ 2. and the deploy contracts (this can take up to 15 minutes):forge 3. script 4. scripts/Deploy.s.sol:Deploy 5. --private-key 6. PRIVATE\_KEY 7. --broadcast 8. --rpc-url 9. ETH\_RPC\_URL 10. forge 11. script 12. scripts/Deploy.s.sol:Deploy 13. --sig 14. 'sync()' 15. --private-key 16. PRIVATE\_KEY 17. --broadcast 18. --rpc-url 19. ETH\_RPC\_URL

## 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
- ~/avail-op-stack-adapter/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/avail-optimism.json
- \
- --deployment-dir
- ../packages/contracts-bedrock/deployments/avail-optimism/
- \
- --outfile.l2
- genesis.json

- \
- --outfile.rollup
- rollup.json
- \
- --l1-rpc=L1\_RPC
- You'll find the newly created genesis.json
- and rollup.json
- in the op-node
- package.
- Generate a jwt.txt
- file, which is crucial for the secure interaction between nodes:
- openssl
- rand
- -hex
- 32
- 
- jwt.txt
- To get op-geth
- ready, move the genesis.json
- and jwt.txt
- files into its directory:
- cp
- genesis.json
- ~/op-geth
- cp
- jwt.txt
- ~/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

Prepare op-geth for running the chain:

1. Navigate to op-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

## Launch and Monitor Nodes

Running op-geth and op-node is essential for every node. op-batcher and op-proposer are exclusive to the sequencer.

Set the following environment variables:

Variable Value SEQ\_KEY Sequencer private key BATCHER\_KEY Batcher private key (minimum 1 ETH) PROPOSER\_KEY Proposer private key L1\_RPC L1 node URL RPC\_KIND L1 server type (e.g., alchemy, quicknode) L2OO\_ADDR L2OutputOracleProxy address

### Running op-geth

To initiate op-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 \ --gcmode=archive \ --nodiscover \ --maxpeers=0 \ --networkid=42069 \ --authrpc.vhosts= "" \ --authrpc.addr=0.0.0.0 \ --authrpc.port=9551 \ --authrpc.jwtsecret=./jwt.txt \ --rollup.disabletxpoolgossip=true 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
```

```
~/avail-op-stack-adapter/op-node ./bin/op-node \ --l2=http://localhost:9551 \ --l2.jwt-secret=./jwt.txt \ --sequencer.enabled \ --sequencer.l1-confs=3 \ --verifier.l1-confs=3 \ --rollup.config=./rollup.json \ --rpc.addr=0.0.0.0 \ --rpc.port=9547 \ --p2p.disable \ --rpc.enable-admin \ --p2p.sequencer.key=SEQ_KEY \ --l1=L1_RPC \ --l1.rpckind=RPC_KIND 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 Goerli ETH for operational continuity.

```
cd
```

```
~/avail-op-stack-adapter/op-batcher ./bin/op-batcher \ --l2-eth-rpc=http://localhost:9545 \ --rollup-rpc=http://localhost:9547 \ --poll-interval=10s \ --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=9548 \ --rpc.enable-admin \ --max-channel-duration=1 \ --l1-eth-rpc=L1_RPC \ -private-key=BATCHER_KEY
```

## Controlling Batcher Costs

Adjust the--max-channel-duration=n setting to balance transaction frequency on L1 and the operational costs of the batcher.

## Running op-proposer

Finally, startup-proposer to propose new state roots:

```
cd
```

```
~/avail-op-stack-adapter/op-proposer ./bin/op-proposer \ --poll-interval=12s \ --rpc.port=9560 \ --rollup-  
rpc=http://localhost:9547 \ --l2oo-address=L2OO_ADDR \ --private-key=PROPOSER_KEY \ --l1-eth-rpc=L1_RPC
```

## Acquire Goerli ETH for Layer 2

To obtain ETH on your Rollup:

1. Go to contracts-bedrock
2. :
3. cd
4. ~/avail-op-stack-adapter/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 Goerli 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 Adapter](#)