How to Use Polygon zkEVM with Avail

Introduction

Embark on setting up your own Polygon zkEVM network, leveraging Avail as the data availability layer. This guide is tailored for deploying on Ethereum's Sepolia testnet and integrating with the Avail Goldberg testnet. To gain a comprehensive understanding of Polygon zkEVM, review thePolygon zkEVM documentation (opens in a new tab) .

In this guide, you will conduct the following:

Prerequisites

Ensure you have installed the following software.

Installation commands are based on Ubuntu 20.04 LTS: Software VersionNode.js (opens in a new tab) Latest LTS Version Git (opens in a new tab) OS Default Golang (opens in a new tab) 1.19 Docker (opens in a new tab) Latest Docker Compose (opens in a new tab) Latest

Install Git

sudo

apt

install

-y

git

Install Node.js (using NVM)

curl

-o-

https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh

|

bash export NVM_DIR = "([-z "{XDG_CONFIG_HOME-}"] &&

printf

%s "{HOME}/.nvm" ||

printf

%s "{XDG_CONFIG_HOME}/nvm")" [-s

"NVM_DIR/nvm.sh"] &&

.

"NVM_DIR/nvm.sh"

This loads nvm

nvm

install

--lts

Install Golang

wget

https://go.dev/dl/go1.19.linux-amd64.tar.gz tar

-C

/usr/local

-xzf

go1.19.linux-amd64.tar.gz echo

"export PATH=PATH:/usr/local/go/bin"

~/.bashrc source

~/.bashrc

Install Docker and Docker Compose

sudo

apt-get

update sudo

apt-get

install

-y

docker-ce

docker-ce-cli

containerd.io

docker-compose-plugin

Hardware Requirements

Both the real and mock provers are compatible exclusively with x86 architectures. They are not designed to operate on ARM architecture machines, including Apple Silicon devices, even within Dockerized environments.

Component Minimum Requirements Recommended Setup Suggested AWS Instance Mock Prover 4-core CPU, 8GB RAM, 50 GB SSD 8-core CPU, 16GB RAM, 60 GB SSD r6a.xlarge Real Prover 96-core CPU, 768GB RAM, 120 GB SSD 96-core CPU, 1000GB RAM, 140 GB SSD r6a.24xlarge Running the Polygon zkEVM solution suite may lead to storage issues, primarily due to excessive Docker logs. To mitigate this, you can customize the Docker daemon's behavior by referring to the first answer[here (opens in a new tab)](). Choose a configuration that best suits your needs.

- For a devnet setup with limited state growth and Docker logs, we recommend a minimum disk size of approximately

50GB.
- If you plan to run a real prover, it's advisable to allocate a minimum disk size of around 120GB for your devnet setup.
- Keep in mind that these recommendations may vary based on your specific use case and requirements.

ADDITIONAL STORAGE CONSIDERATIONS In production environments with a high transaction volume, your storage requirements may increase significantly. It's recommended to utilize an EBS-like data storage solution to ensure scalability, allowing you to add more storage as needed.

Network Details

Before diving into the setup, ensure you have the following network details:

Service URL Explorer http://zkevm-demo.avail.tools/(opens in a new tab) RPC http://zkevm-demo-rpc.avail.tools/(opens in a new tab) Bridge service https://zkevm-demo-bridge.avail.tools/(opens in a new tab)

Launch an Avail-Powered zkEVM

⚠ USAGE DISCLAIMER The prover and verifier components maintain their original security guarantees. However, please note that the data attestation verification during sequencing or any aspect of thevalidium-node related to data availability on Avail has not undergone an audit. Exercise caution when using this program. It is distributed without any warranty, nor an implied warranty of merchantability or fitness for a particular purpose. Please be aware that some aspects of this guide may differ from the original source due to the unique nature of the Avail validium implementation. For zkEVM node-specific configurations and troubleshooting, refer to theofficial Polygon documentation (opens in a new tab).

Deploy the Contracts

1. Clone thevalidium-contracts
2. repository and install dependencies:
3. git
4. clone
5. git@github.com:availproject/validium-contracts.git
6. cd
7. validium-contracts
8. npm
9. i
10. Set up the environment and deployment parameters:
11.
    - Update.env
12.
    - as per.env.example
13.
    - .
14.
    - .env.example
15.
    - Fill indeploy_parameters.json
16.
    - followingdeploy_parameters.json.example
17.
    - :
18.
    -
        - Specify thetrustedSequencer
19.
    -
        - address. This address represents the Sequencer that is responsible for sequencing batches.
20.
    -
        - Define thetrustedAggregator
21.
    -
        - address. This address represents the Aggregator that handles the submission of proofs.
22.
    -
        - Fill in the following fields with the respective addresses that will control the contracts:admin
23.
    -
        - ,zkEVMOwner

24.
    - ,timelockAddress
25.
    - , andinitialZkEVMDeployerOwner
26.
    - .
27.
    - Enter the private key for the deployer in thedeployerPvtKey
28.
    - field.
29.
    - deployment/deploy_parameters.json
30. Execute deployment scripts on the Sepolia network:
31. npx
32. hardhat
33. run
34. --network
35. sepolia
36. deployment/2_deployPolygonZKEVMDeployer.js
37. npx
38. hardhat
39. run
40. --network
41. sepolia
42. deployment/3_deployContracts.js
43. You should generate adeploy_output.json
44. file.
45. Verify the deployed contracts:
46. npx
47. hardhat
48. run
49. --network
50. sepolia
51. deployment/verifyzkEVMDeployer.js
52. npx
53. hardhat
54. run
55. --network
56. sepolia
57. deployment/verifyContracts.js

GENERATING A NEW CONTRACT SUITE To create a fresh set of contracts, you can either employ a new private key or increment the value of thesalt parameter in your configuration. After making this change, simply re-execute the deployment commands to generate the new contract suite.

Deploy the Node

⚠ MOCK PROVER FUNCTIONALITY The Mock Prover does not generate any zero-knowledge proofs. Instead, it simply validates any generated state root as correct. The mock verifier contract operates similarly, accepting all validity proofs without actual verification. 1. Clone thevalidium-node 2. repository for node setup: 3. git 4. clone 5. git@github.com:availproject/validium-node.git 6. cd 7. validium-node 8. Generate a secure account keystore file for Ethereum L1 transactions: 9. docker 10. run 11. --rm 12. hermeznetwork/zkevm-node:latest 13. sh 14. -c 15. "/app/zkevm-node encryptKey --pk=[your private key] --pw=[password to encrypt file] --output=./keystore; cat ./keystore/*" 16. > 17. account.keystore 18. * Replace[your private key] 19. * with your Ethereum L1 account private key. 20. * Replace[password to encrypt file] 21. * with a password used for file encryption. This password must be passed to the Node later via the env variableZKEVM_NODE_ETHERMAN_PRIVATEKEYPASSWORD 22. * . 23. Update configuration files for the node: 24. * Modifytest.avail.config.json 25. * ,test.node.config.toml 26. * , andtest.genesis.config.json 27. * based on the provided example files. 28. * Click to view the avail configuration example 29. * test/config/test.avail.example.config.json 30. * Click to view the node configuration example 31. * test/config/test.node.config.example.toml 32. * Click to view the genesis configuration example 33. * test/config/test.genesis.config.example.json 34. Build the Docker image and launch the node: 35. make 36. build-docker 37. cd 38. test 39. make 40. run

Setup the Prover

1. To switch to the real verifier mode, modify thedeploy_parameters.json
2. file:
3. "realVerifier"
4. :
5. true
6. Utilize the following commands to download and unpack the configuration file:

SIZE: ~70GB+

Accelerate the download process by using a multi-thread downloader like Axel. wget

https://de012a78750e59b808d922b39535e862.s3.eu-west-1.amazonaws.com/v2.0.0-RC4-fork.5.tgz tar

-xzvf

v2.0.0-RC4-fork.5.tgz rm

-rf

config mv

v2.0.0-RC4-fork.5.tgz

validium-node/test/config/prover 1. Ensure thedocker-compose.yml 2. includes proper file mappings for the prover configuration. 3. Modify thetest.prover.config.json 4. to enable actual prover functionality: 5. test.prover.config.json

Configure the Bridge

The zkEVM bridge service is a microservice that simplifies bridging between L1 and L2 by auto-claiming L1 transactions on L2 and generating necessary Merkle proofs. While optional for running a Validium, it enhances the ease of bridging transactions.

The Nomad DA bridge is only operational on Sepolia, limiting validium's data attestation to this chain. Alternatively, you can simulate data attestation and deploy on your preferred blockchain 1. Clone the bridge repository: 2. git 3. clone 4. git@github.com:availproject/validium-bridge-service.git 5. cd 6. bridge 7. Fill inconfig/config.local.toml 8. followingconfig.local.example.toml 9. : 10. Unless you changed the genesis file, the L2 bridge address should remain the same. 11. The address provided by default in the configuration is allocated ETH in the validium test setup for autoclaiming on L2. If a different address is used, it might require ETH. Similarly, in a production setup where ETH is not arbitrarily minted, you will need to manually fund thezkevm-bridge-service 12. autoclaiming account. 13. Parameter 14. Example Value 15. L1URL 16. "http://zkevm-mock-l1-network:8545" 17. GenBlockNumber 18. 1 19. PolygonBridgeAddress 20. "0xff0EE8ea08cEf5cb4322777F5CC3E8A584B8A4A0" 21. PolygonZkEVMGlobalExitRootAddress 22. "0x2279B7A0a67DB372996a5FaB50D91eAA73d2eBe6" 23. L2PolygonBridgeAddresses 24. "0xff0EE8ea08cEf5cb4322777F5CC3E8A584B8A4A0" 25. Click to view the bridge configuration example 26. config/config.local.example.toml 27. Build and run the Docker image using the following commands: 28. make 29. build-docker 30. make 31. run 32. Once the Docker image is running, it serves as a microservice to detect L1 and L2 bridge transactions. You can check if the API is active by accessing the/api 33. endpoint. 34. * Generate Merkle Proofs 35. * : Use the /merkle-proof endpoint to generate the necessary Merkle proofs for bridging transactions. 36. * Additional Endpoints 37. * : The microservice provides other endpoints for various functionalities, such as detecting bridge transactions for specific accounts. 38. * Updating Code 39. * : If you need to modify any part of the code, remember that each change necessitates a new build. To update and rerun the service, execute themake build-docker && make run 40. * commands.

Overview Validium Node _