How to Run a Custom Gas Token Chain

⚠ The Custom Gas Token feature is a Beta feature of the MIT licensed OP Stack. While it has received initial review from core contributors, it is still undergoing testing, and may have bugs or other issues. This guide provides a walkthrough for chain operators who want to run a custom gas token chain. See theCustom Gas Token Explainer for a general overview of this OP Stack feature. An OP Stack chain that uses the custom gas token feature enables an end user to deposit an L1 native ERC20 token into L2 where that asset is minted as the native L2 asset and can be used to pay for gas on L2.

Deploying Your Contracts

- Checkout thev2.0.0-beta.1 of the contracts(opens in a new tab)
- and use the commit to deploy.

Be sure to check out this tag or you will not deploy a chain that uses custom gas token! * Update the deploy config incontracts-bedrock/deploy-config * with new fields:useCustomGasToken * andcustomGasTokenAddress * SetuseCustomGasToken * totrue * . If you setuseCustomGasToken * tofalse * (it defaults this way), then it will use ETH as the gas paying token. * SetcustomGasTokenAddress * to the contract address of an L1 ERC20 token you wish to use as the gas token on your L2. The ERC20 should already be deployed before trying to spin up the custom gas token chain.

The custom gas token being set must meet the following criteria:

- must be a valid ERC-20 token
- the number of decimals on the token MUST be exactly 18
- the name of the token MUST be less than or equal to 32 bytes
- symbol MUST be less than or equal to 32 bytes.
- must not be yield-bearing
- cannot be rebasing or have a transfer fee
- must be transferrable only via a call to the token address itself
- must only be able to set allowance via a call to the token address itself
- must not have a callback on transfer, and more generally a user must not be able to make a transfer to themselves revert
- a user must not be able to make a transfer have unexpected side effects

⚠ You will NOT be able to change the address of the custom gas token after it is set during deployment. * Additionally, ensure that yourbaseFeeVaultWithdrawalNetwork * ,l1FeeVaultWithdrawalNetwork * , andsequencerFeeVaultWithdrawalNetwork * are set to1 * to enable fee withdrawals to L2. Also, set yourbaseFeeVaultRecipient * ,l1FeeVaultRecipient * , andsequencerFeeVaultRecipient * to the L2 address to which you wish to withdraw fees. Note that fee withdrawals to L1 are not currently supported but will be implemented soon. For more details on these values, see theWithdrawal Network * section of the docs. * Deploy the L1 contracts fromcontracts-bedrock * using the following command:

DEPLOYMENT_OUTFILE

deployments/artifact.json \ DEPLOY_CONFIG_PATH =< PATH_TO_MY_DEPLOY_CONFI G

\ forge

script

scripts/Deploy.s.sol:Deploy \ --broadcast

--private-key PRIVATE_KEY \ --rpc-url ETH_RPC_URL * DEPLOYMENT_OUTFILE * is the path to the file at which the L1 contract deployment artifacts are written to after deployment. Foundry has filesystem restrictions for security, so make this file a child of thedeployments * directory. This file will contain key/value pairs of the names and addresses of the deployed contracts. * DEPLOY_CONFIG_PATH * is the path to the file for the deploy config used to deploy

Generating L2 Allocs

Be sure to use the same tag that you used to deploy the L1 contracts. A forge script is used to generate the L2 genesis. It is a requirement that the L1 contracts have been deployed before generating the L2 genesis, since some L1 contract addresses are embedded into the L2 genesis.

CONTRACT_ADDRESSES_PATH

deployments/artifact.json \ DEPLOY_CONFIG_PATH =< PATH_TO_MY_DEPLOY_CONFI G

\ STATE_DUMP_PATH =< PATH_TO_WRITE_L2_ALLOC S

\ forge

script

scripts/L2Genesis.s.sol:L2Genesis \ --sig

'runWithStateDump()'

--chain-id L2_CHAIN_ID To generate L2 Allocs, it is assumed that:

- You have the L1 contracts artifact, specified withDEPLOYMENT_OUTFILE
- CONTRACT_ADDRESSES_PATH
- is the path to the deployment artifact from deploying the L1 contracts in the first step, akaDEPLOYMENT_OUTFILE
- DEPLOY_CONFIG_PATH
- is the path to the file for the deploy config used to deploy
- STATE_DUMP_PATH
- is a path to the generated L2 allocs file, this is the genesis state and is required for the next step.

Generating L2 Genesis

Theop-node is used to generate the final L2 genesis file. It takes the allocs created by the forge script as input for the--l2-allocs flag.

op-node

genesis

l2 \ --l1-rpc ETH_RPC_URL \ --deploy-config

< PATH_TO_MY_DEPLOY_CONFI G

\ --l2-allocs

< PATH_TO_L2_ALLOC S

\ --l1-deployments

< PATH_TO_L1_DEPLOY_ARTIFAC T

\ --outfile.l2

< PATH_TO_WRITE_L2_GENESI S

\ --outfile.rollup

< PATH_TO_WRITE_OP_NODE_CONFI G

Spinning Up Your Infrastructure

Ensure that the end to end system is running.

Validating your deployment

This calls theWETH predeploy which should be considered wrapped native asset in the context of custom gas token. It should return the name of the token prefixed by"Wrapped "

cast

call

--rpc-url L2_ETH_RPC_URL 0x4200000000000000000000000000000000000006

'name()(string)' This calls theL1Block predeploy should returntrue .

cast

call

--rpc-url L2_ETH_RPC_URL 0x4200000000000000000000000000000000000015

'isCustomGasToken()(bool) This calls the L1SystemConfig contract and should returntrue .

cast

call

--rpc-url L1_ETH_RPC_URL < SYSTEM_CONFIG_ADDRES S

'isCustomGasToken()(bool)

Depositing Custom Gas Token into the Chain

- To deposit the custom gas token into the chain, users must use theOptimismPortalProxy.depositERC20Transaction
- method
- Users MUST firstapprove()
- theOptimismPortal
- before they can deposit tokens usingdepositERC20Transaction
- .

function depositERC20Transaction(address _to, uint256 _mint, uint256 _value, uint64 _gasLimit, bool _isCreation, bytes memory _data) public;

Withdrawing Custom Gas Tokens out of the Chain

- To withdraw your native custom gas token from the chain, users must use theL2ToL1MessagePasser.initiateWithdrawal
- method. Proving and finalizing withdrawals is identical to the process on chains that use ETH as the native gas token.

function initiateWithdrawal(address _target, uint256 _gasLimit, bytes memory _data) public payable;

Next Steps

- Additional questions? See the FAQ section in theCustom Gas Token Explainer
- .
- For more detailed info on custom gas tokens, see thespecs(opens in a new tab)
- .
- If you experience any problems, please reach out todeveloper support(opens in a new tab)
- .