Deploy an SVM Warp Route

Outcomes

You will be deploying a Warp Route for an asset of your choice, between two SVM chains with an existing Hyperlane core deployment. At the time of writing this document, supported SVM chains are Solana and Eclipse, but you can find an up-to-date listhere (all chain directory names with acore subdirectory).

info If you want your SVM rollup supported as a core Hyperlane deployment, or are looking to set up an EVM to SVM Warp Route,get in touch !

Warp Route Types

The type of token used determines the Warp Route type, so it's important to understand the different Warp Route contracts available:

- Native
- : Handles the transfer of native gas tokens (e.g. SOL on Solana, ETH on Eclipse).
- Collateral
- : Handles the transfer of existingToken-2022
- orToken
- tokens (the ERC20 equivalent on SVM).
- Synthetic
- : Handles synthetic tokens that are minted and burned as transfers occur through the Warp Route, to represent tokens from their origin chain. The tooling in this guide deploys a new Token-2022 token in this case, whose authority is set to the deployer key.

Here are the common Warp Route setups (you can find more detailshere):

- Native to Synthetic: Lock Native tokens on the origin chain to mint Synthetic ones on the destination. When transferring back, the Synthetic is burned. An example of this is a SOL Warp Route between Solana and Eclipse.
- Collateral to Synthetic: Lock Collateral tokens on the origin chain to mint Synthetic ones on the destination. When transferring back, the Synthetic is burned. An example of this is a USDC Warp Route between Solana and Eclipse.
- Other: Native to Native (such as ETH between Optimism and Arbitrum), as well as Collateral to Collateral, are also possible if the token already exists on both origin and destination chains. Rebalancing liquidity is an important consideration in this case.

Before You Start

Deploying a Warp Route requires there to be a core Hyperlane deployment that is connected (i.e. actively relayed and secured) to the rest of the Hyperlane ecosystem. The core Hyperlane deployments used in this guide are Solana (core artifacts) and Eclipse (core artifacts). You may need to refer to these core artifacts throughout the guide.

Deploy a Sealevel Warp Route

1. Installsolana-cli 1.14.20
2. to build the Warp Route programs. Note that youmust
3. use this version, otherwise deployment may fail.
4. sh
5. -c
6. "
7. (
8. curl
9. -sSfL
10. https://release.solana.com/v1.14.20/install
11.)
12. "
13. Build the Warp Route programs on your machine
14.
    - Clonehyperlane-monorepo
15.
    - Go to./hyperlane-monorepo/rust/sealevel/programs/
16. **starting in rust/sealevel/programs/**

17. cd
18. hyperlane-sealevel-token
19. cargo
20. build-sbf
21. cd
22. ..
23. /hyperlane-sealevel-token-collateral
24. cargo
25. build-sbf
26. cd
27. ..
28. /hyperlane-sealevel-token-native
29. cargo
30. build-sbf
31. To deploy the contracts, installsolana-cli 1.18.18
32. . Note that youmust
33. use this version, otherwise deployment may fail.
34. sh
35. -c
36. "
37. (
38. curl
39. -sSfL
40. https://release.solana.com/v1.18.18/install
41.)
42. "
43. In the monorepo, inrust/sealevel/environments/mainnet3/warp-routes
44. , create a new directory with the name you want your Warp Route deployment to have. For example, the existing SOL Warp Route between Solana and Eclipse lives inrust/sealevel/environments/mainnet3/warp-routes/eclipsesol
45. .
46. If your warp route creates a synthetic token, you can open a PR to thehyperlane-registry
47. with metadata to associate with this token (example PRhere
48.). Thehyperlane-registry
49. also gives your Warp Route visibility within the Hyperlane ecosystem.
50. Configure the parameters of your Warp Route in a JSON file namedtoken-config.json
51. , based on theserde_json
52. serialization of theTokenConfig
53. Rust struct. The value to set for theinterchainGasPaymaster
54. , can be found in thecore deployment artifacts
55. .
56.
    - The example below shows a testnet Native to Synthetic Warp Route that transfers SOL from Solana and mints synthetic SOL on Eclipse. You can also checkthis configuration
57.
    - of a production SOL Warp Route.{
58.
    - "solanatestnet": {
59.
    - "type": "native",
60.
    - "decimals": 9,
61.
    - "interchainGasPaymaster": ""
62.
    - },
63.
    - "eclipsetestnet": {
64.
    - "type": "synthetic",
65.
    - "decimals": 9,
66.
    - "name": "Solana (testnet)",
67.
    - "symbol": "SOL",
68.
    - "uri": ""
69.

    - "interchainGasPaymaster": ""
70.
    - }
71.
    - }
72. Create a Solana private key file. This key pays for the deployment and will be the owner of the deployed programs. An existing funded key can be used if you'd like.
73. solana-keygen new
74. --outfile
75. ./warp-route-deployer-key.json
76. Fund the new key on both networks the Warp Route is being deployed to. The public key should be the same across SVM networks, but do double check with the wallets recommended by each chain, by loading the private key into them.
77.
    - The funding should be enough to cover rent for all accounts related to the Warp Route, pay for transaction fees, and fund theATA
78.
    - payer accounts (more on this below). For reference, the observed rent from one Hyperlane Warp Route account is2.35 SOL
79.
    - on Solana and0.025 ETH
80.
    - on Eclipse, so it's a good idea to fund the key with at least5 SOL
81.
    - /0.05 ETH
82.
    - .
83.
    - To read the public key you just created:solana-keygen pubkey ./warp-route-deployer-key.json
84. Deploy the warp route withwarp-route deploy
85. info
86. Note that since our goal was to make this tooling accessible to developers as soon as possible, it's not as reliable as we would hope. Please get in touch through aGitHub issue
87. or via thedevelopers
88. channel onDiscord
89. if you run into issues.
90.
    - Overview of CLI flags:
91.
    -
        - --warp-route-name
92.
    -
        -
            - should match the directory name picked for the Warp Route earlier
93.
    -
        - --environment
94.
    -
        -
            - keep asmainnet3
95.
    -
        - --environments-dir ../environments
96.
    -
        -
            - keep as../environments
97.
    -
        - --built-so-dir
98.
    -
        -
            - keep as../../target/deploy
99.
    -

- , as it points to the compilation output directory of Warp Route programs

100.
- --token-config-file

101.
- point this to thetoken-config.json

102.
- file created earlier

103.
- --chain-config-file

104.
- keep as../environments/mainnet3/chain-config.json

105.
- , as this file has been pre-populated with chain settings for all Hyperlane-supported chains

106.
- --ata-payer-funding-amount

107.
- this flag specifies by how much to fund the Warp RouteATA

108.
- payer accounts on both chains the deployment happens on. It's expressed in the lowest currency denomination, which means that it's interpreted as Lamports on Solana and Gwei on Eclipse (since it uses ETH as its native currency). In the command below, the value10000000

109.
- works out to0.001

110.
- ETH and0.001

111.
- SOL, which is enough for an initial deployment. ATA payers can always be topped up later, so it’s fine to pick a small value. For reference, every Warp Route transfer costs the ATA payer0.000000001 SOL

112.
- (on Solana) and0.000021 ETH

113.
- (on Eclipse) on the destination chain.

114.
- The script is unlikely to work from the first try due to network congestion and program size, but the script should be idempotent and skip contracts that were already deployed / initialized. Errors likeError: 11 write transactions failed

115.
- orError: Custom: Invalid blockhash

116.
- can always be retried by re-running the command. If retriable errors persist, consider increasing the compute unit pricehere

117.
- .

118.
- For other error types, you may need to close the buffers and programs of your deployer key and redeploy everything from scratch. To display buffers and programs and close them one by one, follow the commands below. Closing programs also helps recover their rent deposit.

119.
- solana program show

120.
    -
        - --programs
121.
    -
        - --keypair
122.
    -
        - ./warp-route-deployer-key.json
123.
    -
        - --url
124.
    -
        - <
125.
    -
        - CHAIN_RPC_URL
126.
    -
        -
127.
    -
        - solana program show
128.
    -
        - --buffers
129.
    -
        - --keypair
130.
    -
        - ./warp-route-deployer-key.json
131.
    -
        - --url
132.
    -
        - <
133.
    -
        - CHAIN_RPC_URL
134.
    -
        -
135.
    -
        - **You'll need to add the `--bypass-warning` flag when closing program accounts (as opposed to closing buffers)**
136.
    -
        - solana program close
137.
    -
        - <
138.
    -
        - YOUR_PROGRAM_ADDRESS
139.
    -
        -

-
  - --url
-
  - <
-
  - CHAIN_RPC_URL
-
  -

- To increase the odds of the deployment succeeding faster, you can set a private RPC url in the--chain-config-file
- passed to the script. (e.g. insolanamainnet.rpcUrls.http
-)
- If deploying a synthetic, the command below will create a new token mint and use the metadata token extension to set the token name, symbol, and metadata json using the fields in the--token-config-file
- file
- Runwarp-route deploy
- **run from rust/sealevel/client**
- cargo
- run --
- -k
- ./warp-route-deployer-key.json warp-route deploy --warp-route-name eclipsesol
- --environment
- mainnet3 --environments-dir
- ..
- /environments --built-so-dir
- ..
- /
- ..
- /target/deploy --token-config-file
- ..
- /environments/mainnet3/warp-routes/eclipsesol/token-config.json --chain-config-file
- ..
- /environments/mainnet3/chain-config.json --ata-payer-funding-amount
- 10000000

Interacting with the Warp Route

1. Let’s query one of the Warp Route programs, getting the program ID from the auto-generatedprogram-ids.json
2. in the directory you created above (wheretoken-config.json
3. also lives). This command prints the Mint Account, Mint Authority, and ATA payer account.
4. **run from rust/sealevel/client**
5. cargo
6. run --
7. -k
8. ./warp-route-deployer-key.json
9. -u
10. <
11. CHAIN_RPC_URL
12.
13. token query --program-id
14. <
15. base58 address from program-ids.json
16.
17. synthetic
18.
    - if deploying a synthetic token, query the Mint Authority account to check out the metadatasolana account
19.
    - <
20.
    - MINT_AUTHORITY
21.
    -
22.
    - --url
23.
    - <
24.
    - CHAIN_RPC_URL
25.
    -
26. Try transferring tokens!
27.
    - You'll need the domain ID of the chain you're sending to, which you can find in the chain'smetadata.yaml
28.
    - entry from thehyperlane-registry
29.
    - .
30. **run from rust/sealevel/client**
31. cargo
32. run --
33. -u
34. <
35. ORIGIN_CHAIN_RPC_URL
36.
37. -k
38. ./warp-route-deployer-key.json token transfer-remote ./warp-route-deployer-key.json
39. <
40. AMOUNT_IN_LOWEST_DENOM
41.
42. <
43. DESTINATION_CHAIN_DOMAIN_ID
44.
45. <

46. RECIPIENT_ADDRESS
47.

48. <
49. WARP_TOKEN_TYPE_ON_ORIGIN_CHAIN: native
50. |
51. synthetic
52. |
53. collateral
54.

55. --program-id
56. <
57. origin chain base58 address from program-ids.json
58.

59. Look for the balance of the recipient on the destination chain, by querying the Mint Account address
60. spl-token balance
61. --owner
62. ./warp-route-deployer-key.json
63. -u
64. <
65. DESTINATION_CHAIN_RPC_URL
66.

67. <
68. MINT_ACCOUNT_ADDRESS
69.

70.
    - The final parameter here is the SPL token ID. So if this is a synthetic warp route you want to check the balance of, you need to use the Mint address from a prior query you made a few steps ago.
71.
    - You can also check out the last tx made to the recipient account in the explorer
72. This guide has made heavy use of thehyperlane-sealevel-client
73. CLI fromhyperlane-monorepo
74. . You may find its various commands useful for configuring the Warp Route, making state queries, sending transfers, and more. Check out the other utilities it provides, in particular those under thetoken
75. subcommand.
76. **run from rust/sealevel/client**
77. cargo
78. run --
79. --help Edit this page Previous How to extend an existing Warp RouteNext How to Go to Production with your Hyperlane Deployment