Deploy Hyperlane with Local Agents

tip This guide is for advanced users who may eventually intend to run Hyperlane agents in a production-like environment. It will cover the basics of how to manually configure and run agents butit is not a production setup guide .

Terminology

The "local chain " is your new chain that you want to deploy Hyperlane onto.

A "remote chain " is a chain with an existing Hyperlane deployment that you want your local chain to send & receive messages to & from.

Overview

There are six steps in this guide:

1. Set up keys
2. that you will use to deploy contracts and run validators and relayers.
3. Deploy contracts
4. to the local chain and to every remote chain with which the local chain will be able to send and receive messages.
5. Run a validator
6. to provide the signatures needed for the Interchain Security Modules you deployed.
7. Run a relayer
8. to send and receive messages between the chains you deployed contracts to.
9. Send a test message
10. to confirm that your relayer is able to deliver messages to and from each pair of chains.
11. Deploy a Warp Route
12. to send token value, not just messages, across chains.

Getting Started

1. Set up keys

There are three keys you must set up and fund.

tip To get started, you can simplify by using the same hexadecimal key for all three roles. Key Role Description Funding Needs Contract Deployer 32 byte hexadecimal private key Funded on all the chains on which we need to deploy contracts. Validator Accounts A list of validator addresses that will sign outbound messages from your local chain. Just one validator works to get started quickly. A small amount so validators can announce the location of their signatures onchain with a one-time transaction Relayer Accounts The single relayer you will operate requires an account on each chain it will deliver messages to & from The relayer must have a balance on all chains it's relaying between. For instructions on how to generate keys, see theagent keys section. Your deployer keymust be a hexadecimal key, while validator and relayer keys can be hexadecimal or AWS KMS.

If deploying on a local network usingFoundry's Anvil , use the following command to fund your newly-generated account. It uses one of the pre-funded private keys to transfer 1 ETH to the address in theYOUR_TARGET_ADDRESS environment variable.

cast send YOUR_TARGET_ADDRESS \ --private-key 0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80 \ --value (cast tw 1)

2. Deploy contracts

Once you have set up deployer, validator, and relayer keys it's time to use the Hyperlane CLI to deploy smart contracts to the local and remote chains.

On the local chain, we will deploy:

- The core contracts, including aMailbox
- that can be used to send and receive messages.

On all chains, we will deploy:

- A Multisig ISM that can be used to verify inbound messages from the other local & remote chains.
- AnInterchainGasPaymaster
- , which can be used to pay our relayer for delivering interchain messages.

- ATestRecipient
- , which we will send messages to, in order to test that everything is working correctly.

Setup

First, install theHyperlane CLI .

npm

install

-g @hyperlane-xyz/cli

Chains

The deployment will require basic information about any chains it will be interacting with. If the target chains are not already in theHyperlane Registry , you must specify chain metadata for them. See theCLI reference for details.

To see what chains are in the currently known, run the following command:

hyperlane registry list To create a chain metadata config for any other chains, run the following command:

hyperlane registry init Or you can define the config manually. See theChainMetadata type for its schema. Seechain-config.yaml for an example.

Core deployment with Multisig ISM

Now the CLI will know how to interact with all your chains, but it will also need to know how to configure your Interchain Security Module (ISM).

To initialize a core config with multisig ISM config, you can define it manually using JSON or YAML (seeexample config here), or create it with the following command:

hyperlane core init --advanced When asked about the ISM type, choosemessageIdMultisigIsm . In the context of this guide, we will be using a 1/1 multisig, so choose a threshold of1 and enter the address of your key.

When asked about the default hook type and required hook type, choosemerkleTreeHook .

Dry Run a Core Deployment

note In order to execute a dry-run, you will need an Anvil node running in a separate terminal instance. To spin up an Anvil node, runanvil . For more on Anvil and installation, checkoutFoundry's Anvil docs . Prior to executing a deployment, you can perform a dry-run via--dry-run or-d to ensure that the deployment will be successful as well as provide analytics on the gas costs of deployment. This will not trigger any real transactions but will simulate a deployment to display a list of required contracts and their respective gas costs.

In addition to validating your upcoming deployment, you can optionally provide the address of the EOA you intend to deploy with via--from-address or-f . This will ensure you have approximately enough funds to complete the deployment.

hyperlane core deploy --dry-run chain1 \ --from-address YOUR_DEPLOYER_ADDRESS

\

(optional) your account address to be impersonated via Anvil; defaults to the HYP_KEY env variable

--registry

\

(optional) path to your primary registry; defaults to the Hyperlane github registry

--overrides

(optional) path to a override registry; defaults to the local ./ path

Deploy

We're now ready to use thecore deploy command to deploy the Hyperlane contracts. To pay for transactions, the command will need the contract deployer key from step 1, which can be provided via theHYP_KEY env variable or as a command argument.

hyperlane core deploy

Verify

Deployment contract artifacts will be written to the configured registries. By default, you'll find the new addresses in the local directory you ran the command from (e.g../chains/chain1/addresses.yaml).

Agent configs

We are now ready to generate an agent config using our deployed contracts that will be used by the agents (Validator and Relayer) in the steps below.

hyperlane registry agent-config --chains chain1 Set theCONFIG_FILES enviroment variable to the path of the agent config generated.

export

CONFIG_FILES

/full/path/to/configs/agent-config.json

3. Run a validator

Validators provide the security for messages sentfrom your chain to remote chains. They're only required when using aMultisig ISM . Learn more about what validators dohere .

Setup directories

First, set theCONFIG_FILES environment variable to the path of the agent config generated in thedeploy contracts step. For example:

export

CONFIG_FILES

/full/path/to/configs/agent-config- { timestamp } .json Next, create a local directory for your validator to write its signatures to. Remember the path, as you will need this when configuring your validator.

danger The validator signatures path will be written on-chain as part of thevalidator announcement transaction .Be careful not to reveal any security-sensitive or personal information!

Pick an informative name specific to the chain you're validating

export

VALIDATOR_SIGNATURES_DIR

/tmp/hyperlane-validator-signatures- < your_chain_name

Create the directory

mkdir

-p

VALIDATOR_SIGNATURES_DIR warning You will not be able to mount anything in/tmp when running the agent via Docker on Mac. To counter this, create a localtmp directory to mount instead.

Create a local tmp directory that can be accessed by docker

mkdir tmp

Pick an informative name specific to the chain you're validating

export

VALIDATOR_SIGNATURES_DIR

tmp/hyperlane-validator-signatures- < your_chain_name

Create the directory

mkdir

-p

VALIDATOR_SIGNATURES_DIR

Configure

There are numerous parameters that validators can be configured with. For this guide, we are concerned with just a handful:

Parameter Description --db Path for writing persistent data to disk. --originChainName Name of the chain being validated (e.g.ethereum). --checkpointSyncer.type Set tolocalStorage for this guide. --checkpointSyncer.path Path to local directory where validator signatures will be written. Same path asVALIDATOR_SIGNATURES_DIR . --validator.key Your validator's hexadecimal private key. info Make sure the validator key corresponds to the address provided when setting up your MultisigIsmConfig. Otherwise, the Multisig ISM you deployed in the previous step will not be able to verify messages sent from your chain. To learn more about all the parameters you can change, read theagent configuration reference .

- Using Docker
- Building from source

Update agent config

Unless you are running Docker on Linux, you will also need to update the agent configuration for your network. This is because Docker does not support thehost network mode on Mac, Windows or Windows Server.

To do this, navigate to the agent-configuration atCONFIG_FILES and replace all instances of "localhost" or "127.0.0.1" in tohost.docker.internal . For example:

... "localnet1": { ... "rpcUrls": [{ // "http": "http://localhost:8545" // "http": "http://127.0.0.1:8545" "http": "http://host.docker.internal:8545" }], ... }, ... Mounting directories

Running with Docker adds an extra layer of complexity because config files need to be accessible from within the Docker container, and validator signatures need to be accessible from outside of the container for the relayer to read. This is so the relayer can construct the metadata required for the message to be successfully validated by the Multisig ISM.

To solve this issue, you can mount directories on your file system into the container. In the arguments below, we:

1. Set theCONFIG_FILES
2. environment variable to a fixed path within the container.
3. Mount the agent config file to this fixed path and making it readonly.
4. Mount the persistent data directory at a fixed path within the container.
5. Mount the validator signatures directory to a fixed path within the container.

.. . -e

CONFIG_FILES

/config/agent-config.json \ --mount

type

bind,source

CONFIG_FILES ,target = /config/agent-config.json,readonly \ --mount

type

bind,source

" (pwd) " /hyperlane_db_validator_ < your_chain_name

> # ,target
>
> /hyperlane_db \ --mount

type

bind,source

" (pwd) " / VALIDATOR_SIGNATURES_DIR ,target = /tmp/validator-signatures \ .. . Hardcoding these paths deduplicates the configuration between docker instances running validators for different origin chains. This makes it easier to pass the right arguments when running the container. See the example below, where the only items to be configured differently for different chains are the chain name and validator key.

.. . ./validator \ --db /hyperlane_db \ --originChainName

< your_chain_name

\ --checkpointSyncer.type localStorage \ --checkpointSyncer.path /tmp/validator-signatures \ --validator.key

< your_validator_key

> .. . Clone and setup

First, clone the Hyperlane monorepo:

git clone git@github.com:hyperlane-xyz/hyperlane-monorepo.git Then follow thesetup instructions in therust directory. This should setuprustup as well as Rosetta 2 if you are on Apple Silicon.

install rustup

curl

--proto

'=https'

--tlsv1.2

-sSf https://sh.rustup.rs |

sh

(apple silicon only) install rosetta 2

softwareupdate --install-rosetta --agree-to-license

Run

- Using Docker
- Building from source

Now that you understand more about configuring validator arguments, pull the latest docker image:

docker pull --platform linux/amd64 gcr.io/abacus-labs-dev/hyperlane-agent:agents-v1.0.0 Before running, ensure that all directories you need to mount are present. This may involve creatinghyperlane_db_validator_ if it does not exist yet.

mkdir

-p hyperlane_db_validator_ < your_chain_name

    Finally, run the validator:

docker run \ -it

\ -e

CONFIG_FILES

/config/agent-config.json \ --mount

type

bind,source

CONFIG_FILES ,target = /config/agent-config.json,readonly \ --mount

type

bind,source

" (pwd) " /hyperlane_db_validator_ < your_chain_name

> # ,target
>
> /hyperlane_db \ --mount

type

bind,source

" (pwd) " / VALIDATOR_SIGNATURES_DIR ,target = /tmp/validator-signatures \ gcr.io/abacus-labs-dev/hyperlane-agent:agents-v1.0.0 \ ./validator \ --db /hyperlane_db \ --originChainName

< your_chain_name

\ --checkpointSyncer.type localStorage \ --checkpointSyncer.path /tmp/validator-signatures \ --validator.key

< your_validator_key

After following the setup instructions, you should now be able to usecargo to run the Validator:

cargo run --release

--bin validator -- \ --db ./hyperlane_db_validator_ < your_chain_name

\ --originChainName

< your_chain_name

\ --checkpointSyncer.type localStorage \ --checkpointSyncer.path

VALIDATOR_SIGNATURES_DIR

\ --validator.key

< your_validator_key

(optional) Run the binary directly You can alternatively build out the agent:

cargo build --release

--bin validator And run the binary directly:

./target/release/validator \ --db ./hyperlane_db_validator_ < your_chain_name

\ --originChainName

< your_chain_name

\ --checkpointSyncer.type localStorage \ --checkpointSyncer.path

VALIDATOR_SIGNATURES_DIR

\ --validator.key

< your_validator_key

For further information, check out theValidators guide .

4. Run a relayer

Relayers deliver interchain messages sent between the local and remote chains. Learn more about what relayers dohere .

You should already have set theCONFIG_FILES environment variable to the path of the agent config generated in theagent configs step. If not, do so now.

export

CONFIG_FILES

/full/path/to/configs/agent-config.json

Configure

There are numerous parameters that validators can be configured with. For this guide, we are concerned with just a handful:

Parameter Description --db Path for writing persistent data to disk. --relayChains Comma separated names of the chains to relay between. E.g.ethereum,polygon,avalanche . --allowLocalCheckpointSyncers Allows the relayer to look for validator signatures on the local filesystem. --defaultSigner.key A hexadecimal private key used to sign transactions for all chains. --metrics-port Optional. The port to expose prometheus metrics on, defaults to9090 . tip Your set of relay chains should include both the origin chain and the destination chain. To learn more about all the parameters you can change, read theagent configuration reference .

- Using Docker
- Building from source

Mounting directories

For the relayer, we provide almost the same arguments to Docker as the validator:

1. Set theCONFIG_FILES
2. environment variable to a fixed path within the container.
3. Mount the agent config file to this fixed path and making itreadonly
4. .
5. Mount the persistent data directory at a fixed path within the container.
6. Mount the validator signatures directory to a fixed path within the container and making itreadonly
7. .

.. . -e

CONFIG_FILES

/config/agent-config.json \ --mount

type

bind,source

CONFIG_FILES ,target = /config/agent-config.json,readonly \ --mount

type

bind,source

" (pwd) " /hyperlane_db_relayer,target = /hyperlane_db \ --mount

type

bind,source

" (pwd) " / VALIDATOR_SIGNATURES_DIR ,target = /tmp/validator-signatures,readonly \ .. . Hardcoding these paths deduplicates the configuration between docker instances running relayers for different sets of chains. This makes it easier to pass the right arguments when running the container. See the example below, where the only items to be configured differently for different chains are the list of chains to relay between and the relayer key. Clone and setup

If you haven't already done so, clone the Hyperlane monorepo and follow thesetup instructions in therust directory.

clone hyperlane monorepo

git clone git@github.com:hyperlane-xyz/hyperlane-monorepo.git

install rustup

curl

--proto

'=https'

--tlsv1.2

-sSf https://sh.rustup.rs |

sh

(apple silicon only) install rosetta 2

softwareupdate --install-rosetta --agree-to-license

Run

- Using Docker
- Building from source

If you haven't already pulled the Docker image, do this now by running:

docker pull --platform linux/amd64 gcr.io/abacus-labs-dev/hyperlane-agent:agents-v1.0.0 Before running, ensure that all directories you need to mount are present. This may involve creatinghyperlane_db_relayer if it does not exist yet.

mkdir

-p hyperlane_db_validator_ < your_chain_name

    Finally, run the relayer:

docker run \ -it

\ -e

CONFIG_FILES

/config/agent-config.json \ --mount

type

bind,source

CONFIG_FILES ,target = /config/agent-config.json,readonly \ --mount

type

bind,source

" (pwd) " /hyperlane_db_relayer,target = /hyperlane_db \ --mount

type

bind,source

" (pwd) " / VALIDATOR_SIGNATURES_DIR ,target = /tmp/validator-signatures,readonly \ gcr.io/abacus-labs-dev/hyperlane-agent:agents-v1.0.0 \ ./relayer \ --db /hyperlane_db \ --relayChains

< chain_1_name

    , < chain_2_name

\ --allowLocalCheckpointSyncers

true

\ --defaultSigner.key

< your_relayer_key

\ After following the setup instructions, you should now be able to usecargo to run the Relayer:

cargo run --release

--bin relayer -- \ --db ./hyperlane_db_relayer \ --relayChains

< chain_1_name

    , < chain_2_name

\ --allowLocalCheckpointSyncers

true

\ --defaultSigner.key

< your_relayer_key

\ --metrics-port 9091 The metrics port is overridden to avoid clashing with the validator.

(optional) Run the binary directly You can alternatively build out the agent:

cargo build --release

--bin relayer And run the binary directly:

./target/release/relayer \ --db ./hyperlane_db_relayer \ --relayChains

< chain_1_name

    , < chain_2_name

\ --allowLocalCheckpointSyncers

true

\ --defaultSigner.key

< your_relayer_key

\ --metrics-port 9091 For further information, check out theRelayer guide .

5. Send test messages

You can check everything is working correctly by sending a test message between pairs of chains. Initiate the message with the CLI.

hyperlane send message --key

PRIVATE_KEY Thesend message command will inform you when the message is dispatched. After a short wait, a confirmation of delivery will be displayed. If the message delivery times out, it is likely that there's an issue with the Validator or Relayer setup in the steps above. To troubleshoot start by looking at the origin chain relayer logs. If you need further help, reach out onDiscord .

6. (Optional) Deploy a Warp Route

To connect tokens using your new Hyperlane deployment, see theWarp Route deployment guide . Edit this page Previous Create your own Hook & ISM