Executor Configuration

Setting your ownExecutor in a LayerZero application means customizing how cross-chain transactions are handled on the destination blockchain.

LayerZero provides a default configuration that offers an Executor, but by configuring your own settings, you can specify which address can execute these transactions and limit the size of messages that can be processed.

CONFIG_TYPE_EXECUTOR

1. maxMessageSize
2. Type
3. : uint32
4. Description
5. : The maximum size of a message that can be sent cross-chain (number of bytes).
6. executor
7. Type
8. : address
9. Description
10. : The address of the executor that is permitted to call thelzReceive
11. function on the destination chain.

Set Config

To use a custom Executor, the OApp owner or assigneddelegate must callsetConfig from the OApp's Endpoint.

info Unlike the ULN configuration, which requires consistent settings for both the sending (source) and receiving (destination) libraries of a chain, the Executor configuration only needs to be set on the send (source) messageLib for remote destination endpoints. This is because the Executor's role is to permissionlessly invokelzReceive once a message is verified. There are no additional permission checks on the destination side, streamlining the process by focusing solely on the source's configuration for outbound message handling.

Here’s how you would encode and set an Executor configuration using Typescript and the ethers library:

Define Parameters

// Using ethers v5 // Define the max message size and the executor address const maxMessageSize =

1024 ;

// This is the maximum message size in bytes const executorAddress =

'0xExecutorAddress' ;

// Address permitted to call lzReceive

// Configuration type const configTypeExecutor =

1 ;

// As defined for CONFIG_TYPE_EXECUTOR

Encode Parameters to Bytes

const executorConfigStructType =

'tuple(uint32 maxMessageSize, address executorAddress)' ; const executorConfigData =

{ maxMessageSize , executorAddress , } ; const executorConfigEncoded = ethers . utils . defaultAbiCoder . encode ([executorConfigStructType] , [executorConfigData] ,) ;

DefineSetConfigParam

const setConfigParamExecutor =

{ eid :

DEST_CHAIN_ENDPOINT_ID ,

// Replace with the destination chain's endpoint ID configType :

CONFIG_TYPE_EXECUTOR ,

// The configuration type for an Executor config : executorConfigEncoded , } ;

CallsetConfig

via the Endpoint

const tx =

await endpointContract . setConfig (oappAddress , messageLibraryAddress ,

[setConfigParamExecutor ,]) ;

await tx . wait ( ) ; To save gas, you can seamlessly configure multiple chain paths in a single function call by including multipleSetConfigParam structs within the array argument ofsetConfig . Just ensure that each struct has a uniqueeid for its remote chain, along with the appropriately encoded configuration parameters for that specific chain.

// Call setConfig with multiple configurations const tx =

await endpointContract . setConfig (oappAddress , messageLibraryAddress ,

[setConfigParamExecutorChainA ,

// Configuration for the first destination chain setConfigParamExecutorChainB ,

// Configuration for the second destination chain // ... add as many configurations as needed]) ;

await tx . wait ( ) ;

Resetting Configurations

Resetting configurations involves callingsetConfig and passing in parameters with nil values. Here's how you can do it:

1. Encode Nil Parameters to Bytes

// Define nil values for Executor configuration const maxMessageSize =

0 ;

// Representing no limit on message size const executorAddress =

'0x0000000000000000000000000000000000000000' ;

// Representing no specific executor address

// Configuration type remains the same for Executor const configTypeExecutor =

1 ;

// As defined for CONFIG_TYPE_EXECUTOR

const executorConfigStructType =

'tuple(uint32 maxMessageSize, address executorAddress)' ; const executorConfigData =

{ maxMessageSize , executorAddress , } ; const executorConfigEncoded = ethers . utils . defaultAbiCoder . encode ([executorConfigStructType] , [executorConfigData] ,) ;

const resetConfigParamExecutor =

{ eid :

DEST_CHAIN_ENDPOINT_ID ,

// Replace with the destination chain's endpoint ID configType : configTypeExecutor , config : executorConfigEncoded , } ; 1. CallsetConfig 2. for Reset

const resetTx =

await endpointContract . setConfig (oappAddress , messageLibraryAddress ,

[resetConfigParamExecutor ,]) ;

await resetTx . wait ( ) ;

Snapshotting Configurations

Snapshotting Executor configurations involves:

- Locking your send library to the default by callingsetSendLibrary
- on the endpoint.
- note
- When configuring your OApp's Executor Config, setting your send library is crucial. If they are not explicitly set when snapshotting configurations, any updates to LayerZero's default message libraries will reset your configurations. This ensures your OApp remains consistent with your security and functionality requirements, safeguarding against unintended changes from default library updates.
- CallinggetConfig
- and passing in the decoded default values intosetConfig
- to lock configuration settings for an OApp.

Locking the Send Library

const sendTx =

await endpointContract . setSendLibrary (oappAddress , eid , sendLibAddress) ; await sendTx . wait ( ) ;

Locking the ULN Config

1. CallgetConfig
2. from the Endpoint

Use thegetConfig function of the endpoint contract to retrieve the current configuration. This function requires the OApp address (_oapp), library address (_lib), endpoint ID (_eid), and configuration type (_configType). It returns the configuration as a bytes array.

Solidity

function

getConfig (address _oapp , address _lib , uint32 _eid , uint32 _configType)

external

view

returns

(bytes

memory config) ; Implementation

const configBytes =

await contract . getConfig (oappAddress , libAddress , eid , configType) ; 1. Decode thegetConfig 2. bytes array

// Define the ABI for the UlnConfig struct const executorConfigAbi =

['tuple(uint32 maxMessageSize, address executorAddress)'] ;

const executorConfigArray = ethers . utils . defaultAbiCoder . decode (executorConfigAbi , configBytes) ; 1. Encode Default Parameters to Bytes

// Configuration type remains the same for Executor const configTypeExecutor =

1 ;

// As defined for CONFIG_TYPE_EXECUTOR const executorConfig = executorConfigArray [0] ;

const executorConfigStructType =

'tuple(uint32 maxMessageSize, address executorAddress)' const executorConfigData =

{ executorConfig . maxMessageSize , executorConfig . executorAddress } const snapshotExecutorConfigEncoded = ethers . utils . defaultAbiCoder . encode ([executorConfigStructType] , [executorConfigData] ,) ;

const snapshotConfigParamExecutor =

{ eid :

DEST_CHAIN_ENDPOINT_ID ,

// Replace with the destination chain's endpoint ID configType : configTypeExecutor , config : snapshotExecutorConfigEncoded , } ; 1. CallsetConfig 2. for Snapshot

const snapshotTx =

await endpointContract . setConfig (oappAddress , messageLibraryAddress ,

[snapshotConfigParamExecutor ,]) ; await snapshotTx . wait ( ) Edit this page

Previous Security Stack (DVNs) Next Deterministic Deployment