Integration with Gelato

TheGelato relay(opens in a new tab) allows developers to execute gasless transactions.

Prerequisites

1. Node.js and npm(opens in a new tab)
2. .
3. Have a Safe configured with threshold equal to 1, where only 1 signature is needed to execute transactions.
4. To use Gelato 1Balance anAPI key(opens in a new tab)
5. is required.

Install dependencies

yarn

add

ethers

@safe-global/relay-kit

@safe-global/protocol-kit

@safe-global/safe-core-sdk-types

Relay Kit options

Currently, the Relay Kit is only compatible with theGelato relay(opens in a new tab) . The Gelato relay can be used in two ways:

1. Gelato 1Balance(opens in a new tab)
2. Gelato SyncFee(opens in a new tab)

Gelato 1Balance

Gelato 1Balance(opens in a new tab) allows you to execute transactions using a prepaid deposit. This can be used to sponsor transactions to other Safes or even to use a deposit on Polygon to pay the fees for a wallet on another chain.

For the 1Balance quickstart tutorial, you will use the Gelato relayer to pay for the gas fees on BNB Chain using the Polygon USDC you have deposited into your Gelato 1Balance account.

For this tutorial you will need a Safe with a threshold of 1 deployed on BNB Chain. You can create one using Safe{Wallet} (opens in a new tab) or theProtocol Kit .

Summary

1. Start with a1/1 Safe on BNB Chain(opens in a new tab)
2. Deposit Polygon USDC into Gelato 1Balance(opens in a new tab)
3. (transaction 0xa5f38(opens in a new tab)
4.)
5. Safe owner0x6Dbd(opens in a new tab)
6. signs aSafe Transaction 0xd94a(opens in a new tab)
7. to send 0.0005 BNB and submits it to Gelato
8. Track the relay request(opens in a new tab)
9. ofGelato Task ID 0x1bf7(opens in a new tab)
10. Transaction 0x814d3(opens in a new tab)
11. is executed on the blockchain

Deposit Polygon USDC into Gelato 1Balance

See:How can I use 1Balance?(opens in a new tab)

Using a Safe account as the Relay App

While using Gelato, you can specify that you only want the relay to allow transactions from specific smart contracts. If one of

those smart contracts is a Safe smart contract, you will need to either verify the contract on a block explorer or get the ABI of the contract implementation (not the ABI of the smart contract address). This is because the Safe smart contracts use theProxy Pattern(opens in a new tab) , so the implementation logic for your smart contract exists on a different address.

Import the packages

import { ethers } from

'ethers' import { GelatoRelayPack } from

'@safe-global/relay-kit' import Safe , { EthersAdapter } from

'@safe-global/protocol-kit' import { MetaTransactionData , MetaTransactionOptions } from

'@safe-global/safe-core-sdk-types'

Initialize the transaction settings

Modify the variables to customize to match your desired transaction settings.

// https://chainlist.org const

RPC_URL

'https://endpoints.omniatech.io/v1/bsc/mainnet/public' const

provider

=

new

ethers .JsonRpcProvider (RPC_URL) const

signer

=

new

ethers .Wallet (process . env . OWNER_1_PRIVATE_KEY ! , provider) const

safeAddress

=

'0x...'

// Safe from which the transaction will be sent

// Any address can be used for destination. In this example, we use vitalik.eth const

destinationAddress

=

'0xd8dA6BF26964aF9D7eEd9e03E53415D37aA96045' const

withdrawAmount

=

ethers .parseUnits ('0.005' ,

'ether') .toString ()

Create the transaction object

// Create a transactions array with one transaction object const

transactions :

MetaTransactionData [] = [{ to : destinationAddress , data :

'0x' , value : withdrawAmount }] const

options :

MetaTransactionOptions

= { isSponsored :

true }

Create the Protocol Kit and Relay Kit instances

const

ethAdapter

=

new

EthersAdapter ({ ethers , signerOrProvider : signer })

const

protocolKit

=

await

Safe .create ({ ethAdapter , safeAddress })

const

relayKit

=

new

GelatoRelayPack ({ apiKey :

process . env . GELATO_RELAY_API_KEY ! , protocolKit })

Prepare the transaction

const

safeTransaction

=

await

relayKit .createRelayedTransaction ({ transactions , options })

const

signedSafeTransaction

=

await

protocolKit .signTransaction (safeTransaction)

Send the transaction to the relay

const

response

=

await

relayKit .executeRelayTransaction (signedSafeTransaction , options)

console .log (Relay Transaction Task ID: https://relay.gelato.digital/tasks/status/ { response .taskId })

Gelato SyncFee

Gelato SyncFee(opens in a new tab) allows you to execute a transaction and pay the gas fees directly with funds in your Safe, even if you don't have ETH or the native blockchain token.

For the SyncFee quickstart tutorial, you will use the Gelato relayer to pay for the gas fees on the BNB Chain using the BNB you hold in your Safe. No need to have funds on your signer.

For this tutorial, you will need a Safe with a threshold of one deployed on the BNB Chain. You can create one usingSafe{Wallet}(opens in a new tab) or theProtocol Kit .

Import the packages

import { ethers } from

'ethers' import { GelatoRelayPack } from

'@safe-global/relay-kit' import Safe , { EthersAdapter } from

'@safe-global/protocol-kit' import { MetaTransactionData } from

'@safe-global/safe-core-sdk-types'

Initialize the transaction settings

Modify the variables to customize to match your desired transaction settings.

// https://chainlist.org const

RPC_URL

'https://endpoints.omniatech.io/v1/bsc/mainnet/public' const

provider

=

new

ethers .JsonRpcProvider (RPC_URL) const

signer

=

new

ethers .Wallet (process . env . OWNER_1_PRIVATE_KEY ! , provider) const

safeAddress

=

'0x...'

// Safe from which the transaction will be sent

// Any address can be used for destination. In this example, we use vitalik.eth const

destinationAddress

=

'0xd8dA6BF26964aF9D7eEd9e03E53415D37aA96045' const

withdrawAmount

=

ethers .parseUnits ('0.005' ,

'ether') .toString ()

Create the transaction object

// Create a transactions array with one transaction object const

transactions :

MetaTransactionData [] = [{ to : destinationAddress , data :

'0x' , value : withdrawAmount }]

Create the Protocol Kit and Relay Kit instances

const

ethAdapter

=

new

EthersAdapter ({ ethers , signerOrProvider : signer })

const

protocolKit

=

await

Safe .create ({ ethAdapter , safeAddress })

const

relayKit

=

new

GelatoRelayPack ({ protocolKit })

Prepare the transaction

const

safeTransaction

=

await

relayKit .createRelayedTransaction ({ transactions })

const

signedSafeTransaction

=

await

protocolKit .signTransaction (safeTransaction)

Send the transaction to the relay

```
const
response
=
await
relayKit .executeRelayTransaction (signedSafeTransaction)
console .log ( Relay Transaction Task ID: https://relay.gelato.digital/tasks/status/ { response .taskId } )
```

Relay Kit Migrating to V2