Simple Relayer

Relayer Engine

The Relayer Engine is a package meant to provide the structure and a starting point for a custom relayer.

With the Relayer Engine, a developer can write specific logic for filtering to receive only the messages they care about.

Once a wormhole message is received, the developer may apply additional logic to parse custom payloads or submit the VAA to one or many destination chains.

To use the Relayer engine, a developer may specify how to relay wormhole messages for their app using an idiomatic express/koa middleware inspired api then let the library handle all the details!

Checkout thequick start example here, or for a more advanced relayer app, see theadvanced example

Quick Start

The source for this example is availablehere

Install Package

First, install therelayer-engine package with your favorite package manager

```

Copy npmi@wormhole-foundation/relayer-engine

```

Start Background Processes

note: These processesmust be running in order for the relayer app below to work

Next, we must start a Spy to listen for available VAAs published on the guardian network as well as a persistence layer, in this case we're using Redis.

More details about the Spy are available in thedocs

Wormhole Network Spy

In order for our Relayer app to receive messages, a local Spy must be running that watches the guardian network. Our relayer app will receive updates from this Spy.

Testnet Spy ```

Copy dockerrun--platform=linux/amd64\ -p7073:7073\ --entrypoint/guardiandghcr.io/wormhole-foundation/guardiand:latest\ spy \ --nodeKey/node.key\ --spyRPC"[::]:7073"\ --network/wormhole/testnet/2/1\ --bootstrap"/dns4/t-guardian-01.nodes.stable.io/udp/8999/quic/p2p/12D3KooWCW3LGUtkCVkHZmVSZHzL3C4WRKWfqAiJPz1NR7dT9Bxh,/dns4/t-guardian-02.nodes.stable.io/udp/8999/quic/p2p/12D3KooWJXA6goBCiWM8ucjzc4jVUBSqL9Rri6UpjHbkMPErz5zK"

Mainnet Spy[](https://docs.wormhole.com/wormhole/quick-start/tutorials/relayer#mainnet-spy)

Copy dockerrun--platform=linux/amd64\ -p7073:7073\ --entrypoint/guardiandghcr.io/wormhole-foundation/guardiand:latest\ spy \ --nodeKey/node.key\ --spyRPC"[::]:7073"\ --network/wormhole/mainnet/2\ --bootstrap"/dns4/wormhole-v2-mainnet-bootstrap.xlabs.xyz/udp/8999/quic/p2p/12D3KooWNQ9tVrcb64tw6bNs2CaNrUGPM7yRrKvBBheQ5yCyPHKC,/dns4/wormhole.mcf.rocks/udp/8999/quic/p2p/12D3KooWDZVv7BhZ8yFLkarNdaSWaB43D
v2-mainnet-bootstrap.staking.fund/udp/8999/quic/p2p/12D3KooWG8obDX9DNi1KUwZNu9xkGwfKqTp2GFwuuHpWZ3nQruS1"

```

Redis Persistence

Note: While we're using Redis here, the persistence layer can be swapped out for some other db by implementing the appropriateinterface .

A Redis instance must also be available to persist job data for fetching VAAs from the Spy.

```

Copy dockerrun--rm-p6379:6379--nameredis-docker-dredis

```

Simple Relayer Code Example

In the following example, we'll:

1. Set up a StandardRelayerApp, passing configuration options for our Relayer
2. Add a filter to capture only those messages our app cares about with a callback to dosomething
3. with the VAA once we've gotten it
4. Start the Relayer app
5.

```

Copy import{ Environment, StandardRelayerApp, StandardRelayerContext, }from"@wormhole-foundation/relayer-engine"; import{ CHAIN_ID_SOLANA }from"@certusone/wormhole-sdk";

(asyncfunctionmain() { // initialize relayer engine app, pass relevant config options constapp=newStandardRelayerApp(Environment.TESTNET, // other app specific config options can be set here for things // like retries, logger, or redis connection settings. { name:"ExampleRelayer", },);

// add a filter with a callback that will be // invoked on finding a VAA that matches the filter app.chain(CHAIN_ID_SOLANA).address(// emitter address on Solana "DZnkkTmCiFWfYTfT41X3Rd1kDgozqzxWaHqsw6W4x2oe", // callback function to invoke on new message async(ctx,next)=>{ constvaa=ctx.vaa; consthash=ctx.sourceTxHash; console.log(Got a VAA with sequence:{vaa.sequence}from with txhash:{hash},); },);

// add and configure any other middleware ..

// start app, blocks until unrecoverable error or process is stopped awaitapp.listen(); })();

```

Explanation

The first meaningful line instantiates theStandardRelayerApp , which is a subclass of theRelayerApp with common defaults.

```

Copy exportclassStandardRelayerApp< ContextTextendsStandardRelayerContext=StandardRelayerContext,

    extendsRelayerApp { // ... constructor(env:Environment,opts:StandardRelayerAppOpts) {

```

The only field we pass in theStandardRelayerAppOpts is the name to help with identifying log messages and reserve a namespace in Redis.

Other StandardRelayerAppOpts options ```

Copy wormholeRpcs?:string[];// List of URLs from which to query missed VAAs concurrency?:number;// How many concurrent requests to make for workflows spyEndpoint?:string;// The hostname and port of our Spy logger?:Logger;// A custom Logger privateKeys?:Partial<{ [kinChainId]: any[]; }>;// A set of keys that can be used to sign and send transactions tokensByChain?:TokensByChain;// The token list we care about workflows?:{ retries:number; };// How many times to retry a given workflow providers?:ProvidersOpts;// Configuration for the default providers fetchSourceTxhash?:boolean;// whether or not to get the original transaction id/hash // Redis config redisClusterEndpoints?:ClusterNode[]; redisCluster?:ClusterOptions; redis?:RedisOptions;
```

``` The next meaningful line in the example adds a filter middleware component. This middleware will cause the Relayer app to request a subscription from the Spy for any VAAs that match the criteria and invoke the callback with the VAA.

If you'd like your program to subscribe to multiple chains and addresses, the same method can be called several times or themultiple helper can be used.

```

Copy app.multiple({

}, myCallback);

```

The last line in the simple example runsawait app.listen() , which will start the relayer engine. Once started, the relayer engine will issue subscription requests to the spy and begin any other workflows (e.g. tracking missed VAAs).

This will run until the process is killed or it encounters an unrecoverable error. If you'd like to shut down the relayer gracefully, callapp.stop() .

## Advanced Example

For a more advanced example that details other middleware and more complex configuration and actions including a built in UI, see the Advanced Tutorial
```