Advanced Relayer Example

This example details a more complex implementation of a Relayer Application. For a simple example see thisexample

The source for this example is availablehere

Setup

note: In order to run the Spy and Redis for this tutorial, you must havedocker installed.

Get the code

Clone the repository,cd into the directory, and install the requirements.

```

Copy gitclonehttps://github.com/wormhole-foundation/relayer-engine.git cdrelayer-engine/examples/advanced/ npmi

```

Run it

Start the background services

Start the Spy to subscribe to gossiped messages on the Guardian network.

```

Copy npmruntestnet-spy

```

In another CLI window, start Redis. For this application, Redis is used as the persistence layer to keep track of VAAs we've seen.

```

Copy npmrunredis

```

Start the relayer

Once the background processes are running, we can start the relayer. This will subscribe to relevant messages from the Spy and track VAAs, taking some action when one is received.

```

Copy npmrunstart

```

Code Walkthrough

Context

The first meaningful line is a Type declaration for theContext we want to provide our Relayer app.

```

Copy exporttypeMyRelayerContext=LoggingContext& StorageContext& SourceTxContext& TokenBridgeContext& StagingAreaContext& WalletContext;

```

This type, which we later use to parameterize the genericRelayerApp , specifies the union ofContext objects that are available to theRelayerApp .

Because theContext object is passed to the callback for processors, providing a type parameterized type definition ensures the appropriate fields are available within the callback on theContext object.

App Creation

Next we instantiate aRelayerApp , passing ourContext type to parameterize it.

```

Copy constapp=newRelayerApp(Environment.TESTNET);

```

For this example we've defined a class,ApiController , to provide methods we'll pass to our processor callbacks. Note that thectx argument type matches theContext type we defined.

```

Copy exportclassApiController{ processFundsTransfer=async(ctx:MyRelayerContext,next:Next)=>{ // ... }; }

```

This is not required but a pattern like this helps organize the codebase as theRelayerApp grows.

We instantiate our controller class and begin to configure our application by passing the Spy URL, storage layer, and logger.

```

Copy constfundsCtrl=newApiController(); constnamespace="simple-relayer-example"; conststore=newRedisStorage({ attempts:3, namespace,// used for redis key namespace queueName:"relays", });

// ...

app.spy("localhost:7073"); app.useStorage(store); app.logger(rootLogger);

```

Middleware

With our app configured, we can begin to addMiddleware .

Middleware is the term for the functional components we wish to apply to each VAA received.

TheRelayerApp defines ause method which accepts one or moreMiddleware instances, also parameterized with aContext type.

```

Copy use(...middleware: Middleware[]|ErrorMiddleware[])

```

By passing theuse method an instance of someMiddleware , we add it to the pipeline of handlers invoked by theRelayerApp .

Note that the order theMiddleware is added here matters since a VAA is passed through each in the same order.

```

Copy // we want an instance of a logger available on the context app.use(logging(rootLogger)); // we want to check for any missed VAAs if we receive out of order sequence ids app.use(missedVaas(app,{ namespace:"simple",logger:rootLogger })); // we want to apply the chain specific providers to the context passed downstream app.use(providers()); // enrich the context with details about the token bridge app.use(tokenBridgeContracts()); // ensure we use redis safely in a concurrent environment app.use(stagingArea()); // make sure we have the source tx hash app.use(sourceTx());

```

Subscriptions

With ourMiddleware setup, we can configure a subscription to receive only the VAAs we care about.

Here we set up a subscription request to receive VAAs that originated from Solana and were emitted by the addressDZnkkTmCiFWfYTfT41X3Rd1kDgozqzxWaHqsw6W4x2oe .

On receipt of a VAA that matches this filter, thefundsCtrl.processFundsTransfer callback is invoked with an instance of theContext object that has already been passed through theMiddleware we set up before.

```

Copy app .chain(CHAIN_ID_SOLANA) .address( "DZnkkTmCiFWfYTfT41X3Rd1kDgozqzxWaHqsw6W4x2oe", fundsCtrl.processFundsTransfer, );

```

To subscribe to more chains or addresses, this pattern can be repeated or themultiple method can be called with an object ofChainId toAddress

```

Copy app.multiple( {

}, fundsCtrl.processFundsTransfer, );

```

Error Handling

The lastMiddleware we apply is an error handler, which will be called any time an upstreamMiddleware component throws an error.

Note that there are 3 arguments to this function which hints to theRelayerApp that it should be used to process errors.

```

Copy app.use(async(err,ctx,next)=>{ ctx.logger.error("error middleware triggered"); });

```

Start listening

Finally, callingapp.listen() will start theRelayerApp , issuing subscription requests and handling VAAs as we've configured it.

Thelisten method is async andawait -ing it will block until the program dies from an unrecoverable error, the process is killed, or the app is stopped withapp.stop() .

Bonus UI

For this example, we've provided a default UI usingkoa .

When the program is running, you may open a browser tohttp://localhost:3000/ui to see details about the running application.

Going further

The included default functionality may be insufficient for your use case.

If you'd like to apply some specific intermediate processing steps, consider implementing some customMiddleware . Be sure to include the appropriateContext in theRelayerApp type parameterization for any fields you wish to have added to theContext object passed to downstreamMiddleware .

If you'd prefer a storage layer besides redis, simply implement thestorage interface.

Wormhole integration complete?

Let us know so we can list your project in our ecosystem directory and introduce you to our global, multichain community!

Reach out now!

Last updated1 month ago

On this page * Setup * Get the code * Run it * Start the background services * Start the relayer * Code Walkthrough * Context * App Creation * Middleware * Subscriptions * Error Handling * Start listening * Bonus UI * Going further

Was this helpful? Edit on GitHub