Drizzle and Contract Events¶

Drizzle is ideal for synchronizing contract state with a user interface, but as dapps grow in complexity we foresee growing demand for coordination with off-chain services.

The code in this guide may be small, but it opens up a powerful mechanism to enable that coordination. Imagine your dapp needs to send a message whenever a contract event is generated. Lets see how to make this happen.

This tutorial demonstrates how to use Drizzle to subscribe to smart contract events.

The finished dapp.

Getting Started¶

We'll usereact-toastify to alert the user whenever a SimpleStorage contract event is emitted. We have to declare acomponent and invoketoast.success() when an event is detected. We'll touchMyComponent and the event reducer respectively.

A Complete example is available at the followingrepo .

Prerequisite : You should be familiar with Truffle, Drizzle, React and Redux. If you need an introduction please consult the following resources:1. Truffle Quickstart 2. Getting Started with Drizzle and React 3. Tutorial: Intro to React 4. Redux Basic Tutorial 5. Redux Saga 6. Redux Middleware

Unbox Drizzle¶

Let's usetruffle unbox to bootstrap a project and then wire up a contract event to a display component by creating a reducer and hook it up to Drizzle'sEVENT_FIRED action.

Note : More Drizzle actions are listed in ourDrizzle Actions documentation .

First create an empty directory, navigate to it, andunbox Drizzle.

// In the project directory... truffle unbox drizzle Now let's start thetruffle develop console (which runs a Ganache test blockchain in the background):

truffle develop Finally, in our Truffle develop console, let's compile and migrate our smart contracts.

truffle(develop)

> compile# some output here... truffle(develop)
>
> migrate Now that we have a test chain running and our smart contracts deployed, let's add a toast notification to the UI.

Listen for Contract Events¶

We want to listen for theSimpleStorage contract'sStorageSet event and show a toast notification once it fires.

The front end code is located under theapp folder. Lets add the notification libraryreact-toastify to simulate an external interaction From the project directory:

cd

app npm install react-toastify For the sake of simplicity, we will work in one file,./app/src/middleware/index.js .

mkdir ./src/middleware touch ./src/middleware/index.js In./app/src/middleware/index.js , importEventActions andgenerateStore from Drizzle as well as toast fromreact-toastify , anddrizzleOptions .

// ./app/src/middleware/index.js import

{

generateStore ,

EventActions

}

from

'@drizzle/store' import

drizzleOptions

from

'../drizzleOptions' import

{

toast

}

from

'react-toastify' The actionEventActions.EVENT_FIRED is emitted whenever a contract event is detected in a Block. We will gain access to it by registering a custom middleware with the Redux store. As you may know, Redux middleware comprises a set of functions, executed in a sequence, that process each dispatched action before passing it to Reducers. Add the following:

const

contractEventNotifier

=

store

=>

next

=>

action

=>

{

if

(action . type

===

EventActions . EVENT_FIRED)

{

const

contract

=

action . name

const

contractEvent

=

action . event . event

const

message

=

action . event . returnValues . _message

const

display

=

{ contract } ({ contractEvent }): { message }

toast . success (display ,

{

position :

toast . POSITION . TOP_RIGHT

})

}

return

next (action) } Now lets register this middleware with Drizzle.generateStore will return a Redux store that you can use anywhere you can use a store. We will export it to be used byDrizzleProvider . Add the following:

const

appMiddlewares

=

[

contractEventNotifier

] export

default

generateStore ({

drizzleOptions ,

appMiddlewares ,

disableReduxDevTools :

false

// enable ReduxDevTools! }) Your completed./app/src/middleware/index.js file should look something like this:

import

{

generateStore ,

EventActions

}

from

'@drizzle/store' import

drizzleOptions

from

'../drizzleOptions' import

{

toast

}

from

'react-toastify' const

contractEventNotifier

=

store

=>

next

=>

action

=>

{

if

(action . type

===

EventActions . EVENT_FIRED)

{

const

contract

=

action . name

const

contractEvent

=

action . event . event

const

message

=

action . event . returnValues . _message

const

display

=

{ contract } ({ contractEvent }): { message }

toast . success (display ,

{

position :

toast . POSITION . TOP_RIGHT

})

}

return

next (action) } const

appMiddlewares

=

[

contractEventNotifier

] export

default

generateStore ({

drizzleOptions ,

appMiddlewares ,

disableReduxDevTools :

false

// enable ReduxDevTools! })

Connect the Store¶

The first thing we need to do is send thestore as a prop toDrizzleProvider . Add the following to./app/src/App.js :

// ./app/src/App.js ... import

store

from

'./middleware' ... const

drizzle

=

new

Drizzle (drizzleOptions ,

store); ... Your completedApp.js file should looks something like this:

import

React

from

"react" ; import

{

DrizzleContext

}

from

"@drizzle/react-plugin" ; import

{

Drizzle

}

from

"@drizzle/store" ; import

drizzleOptions

from

"./drizzleOptions" ; import

MyComponent

from

"./MyComponent" ; import

"./App.css" ; import

store

from

"./middleware" ; const

drizzle

=

new

Drizzle (drizzleOptions ,

store); const

App

=

()

=>

{

return

(

< DrizzleContext . Provider

drizzle

{ drizzle }

< DrizzleContext . Consumer

{ drizzleContext

=>

{

const

{

drizzle ,

drizzleState ,

initialized

}

=

drizzleContext ;

if

(! initialized)

{

return

"Loading..."

}

return

(

< MyComponent

drizzle

{ drizzle }

drizzleState

{ drizzleState }

/>

)

}}

< /DrizzleContext.Consumer>

< /DrizzleContext.Provider>

); } export

default

App ;

Hook up Display¶

Next we need to modify./app/src/MyComponent.js to importReactToastify.css and configureToastContainer

... import

{

ToastContainer

}

from

'react-toastify' import

'react-toastify/dist/ReactToastify.css' ... export

default

({

accounts

})

=>

(

< div

className

"App"

< ToastContainer

/>

...

< /div> Your completedMyComponent.js should look something like:

import

React

from

"react" ; import

{

newContextComponents

}

from

"@drizzle/react-components" ; import

logo

from

"./logo.png" ; import

{

ToastContainer

}

from

'react-toastify' ; import

'react-toastify/dist/ReactToastify.css' ; const

{

AccountData ,

ContractData ,

ContractForm

}

=

newContextComponents ; export

default

({

drizzle ,

drizzleState ,

accounts

})

=>

{

// destructure drizzle and drizzleState from props

return

(

< div

className

"App"

< ToastContainer

/>

< div

< img

src

{ logo }

alt

"drizzle-logo"

/>

< h1

Drizzle

Examples < /h1>

< p

Examples

of

how

to

get

started

with

Drizzle

in

various

situations .

< /p>

< /div>

< div

className

"section"

< h2

Active

Account < /h2>

< AccountData

drizzle

{ drizzle }

drizzleState

{ drizzleState }

accountIndex

{ 0 }

units

"ether"

precision

{ 3 }

/>

< /div>

< div

className

"section"

< h2

SimpleStorage < /h2>

< p

This

shows

a

simple

ContractData

component

with

no

arguments ,

along

with

a

form

to

set

its

value .

< /p>

< p

< strong

Stored

Value :

< /strong>

< ContractData

drizzle

{ drizzle }

drizzleState

{ drizzleState }

contract

"SimpleStorage"

method

"storedData"

/>

< /p>

< ContractForm

drizzle

{ drizzle }

contract

"SimpleStorage"

method

"set"

/>

< /div>

...

< /div>

); };

A Quick Test¶

- Things often go south during development so a pretest check is in order.
- 1. MetaMask should NOT be on Mainnet! Do not run this if you're on Mainnet!
- 1. Is MetaMask listening on the correct port defined above? Metamask should have ETH funds. Something is amiss if it doesn't.
- 1. Are the smart contracts deployed from the correct directory?
- Fire up the app.
- npm
- run
- start
- Change SimpleStorage'sstored Value
- You'll be rewarded with a toast notification when the transaction is completed.

The dapp is now a consumer of Drizzle'sEVENT_FIRED action item and can coordinate with other services to implement its business logic.

More Drizzle actions are listed in ourDrizzle Actions documentation .