Truffle Suite

Archived: This tutorial has been archived and may not work as expected; versions are out of date, methods and workflows may have changed. We leave these up for historical context and for any universally useful information contained. Use at your own risk!

Webpack is a powerful module bundler that helps turn your code into static assets you can deploy to the web. With theTruffle Solidity loader , we'll show you how you can use Webpack with your existing Truffle project.

Intended Audience¶

This tutorial is for Truffle users who are familiar with Webpack already. Since Webpack requires considerable knowledge to use effectively, we recommend checking out theWebpack documentation as well as theTruffle + Webpack Demo application before proceeding.

Getting Started¶

Navigate to your project in the command line, and then install the required dependencies. These dependencies are in addition to any other Webpack-related dependencies your project might have.

npm install webpack webpack-dev-server truffle-solidity-loader --save-dev

Set up Your Webpack Configuration¶

You should already have a webpack configuration for your project. If you don't, an example configuration file is provided at the bottom of this tutorial. To enable Solidity support, add thetruffle-solidity-loader as a plugin within your project:

// your webpack configuration... module :

{

loaders :

[

// ...,

{

test :

/.sol/ ,

loader :

'truffle-solidity'

}

] }

Remove Default Build Configuration¶

Since you're using Webpack to build your frontend, you won't want to use the default Truffle build process as the two processes will conflict. To ensure there are no issues if you or your teammates runtruffle build , go ahead and delete thebuild configuration in yourtruffle.js file, if it exists.

module . exports

=

{

build :

{

// Delete this.

// ... // and this.

},

// and this too.

rpc :

{

host :

"127.0.0.1" ,

port :

8545

} }

Bootstrapping Your Application¶

Since you're bundling the frontend yourself and not using Truffle's default build process, certain "magic" provided by Truffle needs to be handled in order to get your application running properly. This includes:

1. Detecting theweb3
2. object provided to you by the user's Ethereum client, which as of this writing will likely be eitherMetamask
3. orMist
4. .
5. Provisioning your contract abstractions so they can communicate with your user's Ethereum client.

Detecting Web3¶

In the past, Truffle bootstrapped applications to look for an open Ethereum node running athttp://localhost:8545 , but this has proven to be insecure for the user as well as highly unlikely in a real world scenario. Instead, users are most likely to use Metamask or Mist to interact with web applications on the Ethereum network, and you should build your application to support that. Both Metamask and Mist inject aweb3 object when the page is loading so you can hook into their transaction signing processes. Detecting thisweb3 object easy:

var

Web3

=

require ("web3"); window . addEventListener ('load' ,

function ()

{

// Supports Metamask and Mist, and other wallets that provide 'web3'.

if

(typeof

web3

!==

'undefined')

{

// Use the Mist/wallet provider.

window . web3

=

new

Web3 (web3 . currentProvider);

}

else

{

// No web3 detected. Show an error to the user or use Infura: https://infura.io/

} }); Notice we only use the wallet's provider and not the wholeweb3 object provided to us. This ensures our application is not dependent on the wallet's version of Web3, and reduces the surface area in which version errors might occur.

Provisioning Your Contract Abstractions¶

Provisioning your contract abstractions requires two things: 1) that you include your contracts in your project, and 2) that you set up the Javascript abstractions so that they're correctly talking to the Ethereum client via Web3.

When including contracts in your application, the Solidity Loader will resolve all.sol files to their associated contract abstraction, so importing your contracts is simply a matter of performing one of the following within your frontend:

// ES5: var

MyContract

=

require ("contracts/MyContract.sol"); // ES6: import

MyContract

from

"contracts/MyContract.sol" ; After making sure your contract is imported, you now need to hook up theweb3 object you detected above with the contract abstraction provided:

MyContract . setProvider (window . web3 . currentProvider); Finally, you can use the contract abstraction as described within theTruffle documentation .

You can prevent having to perform this provisioning more than once per contract dependency by either setting your contract abstractions to the global object, likewindow , and provisioning them once when your application loads; or using something like theProvidePlugin to ensure your contract abstractions are available to every file within the bundle (recommended).

Running Webpack¶

From here you can run webpack like normal, via the normalwebpack command or thewebpack-dev-server , if you have those installed globally on your machine:

webpack

webpack-dev-server --hot Alternatively, you can edit your project'spackage.json file to run Wepback from the versions you installed at the beginning of this tutorial:

"scripts" :

{

"build" :

"webpack" ,

"dev" :

"webpack-dev-server --hot" }

Example Webpack Config¶

Here's a very simple Webpack configuration file that uses the Truffle Solidity Loader as well as other plugins. This configration file exists within a React application where its source files exist within the./app directory, and CSS is pulled out using thecopy-webpack-plugin . Of course, your dapp's layout and structure will be different, but this should give you a good example of how the Solidity Loader fits within a simple application.

```
var
webpack
=
require ( "webpack" ); var
CopyWebpackPlugin
=
require ( 'copy-webpack-plugin' ); var
ExtractTextPlugin
=
require ( "extract-text-webpack-plugin" ); module . exports
=
{
entry :
'./app/javascripts/app.jsx' ,
output :
{
path :
"./build" ,
filename :
'app.js'
},
module :
{
loaders :
[
{
test :
/.(js|jsx|es6)/ ,
exclude :
/node_modules/ ,
loader :
"babel-loader" },
{
test :
/.scss/i ,
loader :
ExtractTextPlugin . extract ([ "css" ,
```

```
"sass" ])},
{
test :
/.json/i ,
loader :
"json-loader" },
{
test :
/.sol/ ,
loader :
'truffle-solidity'
}
]
},
plugins :
[
new
CopyWebpackPlugin ([
{
from :
'./app/index.html' ,
to :
"index.html"
},
{
from :
'./app/images' ,
to :
"images"
},
{
from :
'./app/fonts' ,
to :
"fonts"
}
]),
```

```
new
ExtractTextPlugin ( "app.css" )
],
devServer :
{
stats :
'errors-only' ,
} };
```