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Archived: This tutorial has been archived and may not work as expected; versions are out of date, methods and workflows may have changed.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!

Smart contracts deployed to the Ethereum mainnet can deal with real money, so having our Solidity code free from errors and highly secure is essential.

Zeppelin Solutions , a smart contract auditing service, has recognized this need. Using their experience, they've put together a set of vetted smart contracts calledOpenZeppelin .

We can use and extend these contracts to create more secure dapps in less time. OpenZeppelin comes with a wide array of smart contracts for various important functions (see them all here), but today we'll be focusing on their token contracts. Specifically, we'll be extending theirStandardToken.sol contract to create our ownERC20 -compliant token.

Requirements¶

This tutorial expects you to have some knowledge of Truffle, Ethereum, and Solidity. If you haven't gone through ourEthereum overview and ourPet Shop tutorial yet, those would be great places to start.

For even more information, please see the following links:

- Truffle documentation
- Ethereum
- Solidity documentation

We will primarily be using the command line for this tutorial, so please ensure you have basic familiarity with your operating system's terminal.

Overview¶

In this tutorial we will be covering:

- Unboxing the front-end application
- Creating the "TutorialToken" smart contract
- Compiling and deploying the smart contract
- Interacting with the new token

Unboxing the front-end application¶

In this tutorial, we are focusing on smart contract creation. To that end, we've created the front-end for you in the form of a Truffle Box.

1. On a terminal, create a project directory and navigate to it:

mkdir oz-workspacecd

oz-workspace 1. Unbox thetutorialtoken 2. Truffle Box. This will give us our project template.

truffle unbox tutorialtoken 1. Next, we'll install OpenZeppelin. The most recent version of OpenZeppelin can be found as an npm package.

npm install openzeppelin-solidity

Creating the "TutorialToken" smart contract¶

With our front-end taken care of, we can focus on theTutorialToken contract.

1. In thecontracts/
2. directory of your Truffle Box, create the fileTutorialToken.sol
3. and add the following contents:

pragma solidity

^ 0.4.24 ; import

"openzeppelin-solidity/contracts/token/ERC20/ERC20.sol" ; contract

TutorialToken

is

ERC20 { } Things to notice:

- Beyond the standard smart contract setup, we import theStandardToken.sol
- contract and declare ourTutorialToken
- .
- We useis
- to inherit from theStandardToken
- contract. Our contract will inherit all variables and functions from theStandardToken
- contract. Inherited functions and variables can be overwritten by redeclaring them in the new contract.
- To set our own parameters for the token, we'll be declaring our own name, symbol, and other details. Add the following content block to the contract (between the curly braces):

string

public name

=

"TutorialToken" ; string

public symbol

=

"TT" ; uint8

public decimals

=

2 ; uint

public INITIAL_SUPPLY

=

12000 ; Things to notice:

- Thename
- andsymbol
- variables give our token a unique identity.
- Thedecimals
- variable determines the degree to which this token can be subdivided. For our example we went with 2 decimal places, similar to dollars and cents.
- TheINITIAL_SUPPLY
- variable determines the number of tokens created when this contract is deployed. In this case, the number is arbitrary.
- To finish up our contract, we'll create a constructor function to mint with thetotalSupply
- equal to our declaredINITIAL_SUPPLY
- and give the entire supply to the deploying account's address. Add this block below the content added in the previous step:

constructor ()

public

{

_mint(msg.sender ,

INITIAL_SUPPLY); } Using less than 15 lines of hand-coded Solidity, we've created our own Ethereum token!

Compiling and deploying the smart contract¶

1. In themigrations/
2. directory, create the file2_deploy_contracts.js
3. and add the following content:

var

TutorialToken

=

artifacts . require ("TutorialToken"); module . exports

=

function (deployer)

{

deployer . deploy (TutorialToken); }; Theimport statement within ourTutorialToken contract will be automatically handled by the compiler, along with any subsequent imports withinStandardToken .

1. Now we are ready to compile and deploy your contract to the blockchain. For this tutorial, we will useGanache
2. , a personal blockchain for Ethereum development you can use to deploy contracts, develop applications, and run tests. If you haven't already,download Ganache
3. and double click the icon to launch the application. This will generate a blockchain running locally on port 7545.

Note: Read more about Ganache in theGanache documentation.

1. With our blockchain launched, head back to your terminal. Inside your project, run the following command to compile the contract:

truffle compile Note : If you're on Windows and encountering problems running this command, please see the documentation on resolving naming conflicts on Windows.

1. Once the compile has completed, deploy the contract to the blockchain:

truffle migrate You will see output that looks similar to this:

Using network 'development' .

Running migration: 1_initial_migration.js Deploying Migrations... ... 0xa4470beb31f490e26b9a8b0d677cb7107ae5ef5bf40f8ee59fe040d35ca4f598 Migrations: 0x8cdaf0cd259887258bc13a92c0a6da92698644c0 Saving successful migration to network... ... 0xd7bc86d31bee32fa3988f1c1eabce403a1b5d570340a3a9cdba53a472ee8c956 Saving artifacts... Running migration: 2_deploy_contracts.js Deploying TutorialToken... ... 0xcc01dcbe77f79cf4c21e5642bfee50a6db78b7d6dedb9f8b363ec4110e92436d TutorialToken: 0x345ca3e014aaf5dca488057592ee47305d9b3e10 Saving successful migration to network... ... 0xf36163615f41ef7ed8f4a8f192149a0bf633fe1a2398ce001bf44c43dc7bdda0 Saving artifacts... Ganache will also list these transactions as well.

Interacting with the new token¶

For this portion of the tutorial, we recommend usingMetaMask , a browser extension for Chrome and Firefox. It will allow you to switch between accounts quickly, perfect for testing the ability to transfer our newly created tokens. OurPet Shop tutorial has more information aboutconfiguring MetaMask .

You will want to enter the mnemonic displayed in Ganache into MetaMask, and make sure that MetaMask is listening to the Custom RPChttp://127.0.0.1:7545 .

Warning: Do not use the Main Network in MetaMask. If you send ether to any account generated from Ganache's default mnemonic, you will lose it all!

1. Still in your terminal, run a local web server containing the front-end application:

npm run dev A browser window should automatically open with the interface below:

Our basic dapp shows the TutorialToken balance of the selected account in MetaMask.

1. Now we'll transfer some TutorialToken tokens to a different account. Ganache, when launched, lists 10 accounts. The first account has the token balance. Pick one of the other accounts (we recommend the second account) and enter it in the "Address" box, and also enter2000

2. in the "Amount" field.

3. Click "Transfer" to initiate the token transfer. MetaMask will intercept the transfer request and display a confirmation. Note that no ether is changing hands, except for the gas used to pay for the transaction.

4. Click "Submit" and the transfer will proceed. If all goes well, you will see a window saying "Transfer successful". You will also see a record of the transaction in MetaMask, and a new transaction will be displayed at the top of the "Transactions" section in Ganache.

5. Still in MetaMask, switch from the first account to the second one (you may need to select "Create an account" if only one account is in the list.)

6. Now refresh the app in your browser. It will be connected to the currently selected account in MetaMask, and display the amount of tokens (in this case, 2000 TT). This shows that the transfer did in fact succeed.

7. Try sending different amount of tokens to different accounts to practice how our dapp (and MetaMask) interacts with the network.

We at Truffle are excited to see companies like Zeppelin Solutions contributing to the standardization and increased security of smart contracts. With OpenZeppelin's contracts and Truffle's tools, you have everything you need to start creating industry-standard distributed applications.

Happy coding!