Using Truffle to interact with Chainlink Smart Contracts

Chainlink is one of the leading blockchain projects building the very first decentralized oracle network that provides external data to smart contracts. Truffle helps smart contract developers by providing a powerful framework to create, test, and deploy smart contracts. In Chainlink's blogpost , theirChainlink Truffle Box has gotten some significant upgrades! If you were fortunate enough to attend Trufflecon 2019, then you have already seen these improvements in action, thanks to Chainlink's presentation by Thomas Hodges. To start developing your very own Chainlink smart contracts, you can grab Chainlink's Truffle Box by following the guidehere .

This article will showcase these brand new additions, starting withTruffle Scripts . Scripts create a much easier approach to interact with our contracts, allowing us to fund a contract with LINK, request data from it, and read the contract's data with only three commands. To demonstrate this functionality, we will first need to set up our environment variables: MNEMONIC and RPC_URL.

Working with blockchain technology requires a wallet, with one of the most popular choices beingMetaMask . Within minutes, you can have your very own Ethereum wallet that allows you to easily interact with various dApps. During account generation, you will be shown yoursecret 12-word seed phrase, known as your mnemonic. (Do not share this with anyone; if someone knows yourmnemonic they WILL have access to your addresses and private keys). Be sure to store this mnemonic somewhere safe, as you will need it later to work with our example.

Furthermore, you will also need to fund your MetaMask wallet withtest LINK andtest ETH , in order to power transactions on the Ropsten Ethereum Testnet.

Note: When funding your account from a test faucet, make sure you switch from the default Main Ethereum Network to the Ropsten Test Network, to properly see your funds. You can follow any transactions made on the Ropsten Testnet via Etherscan.

Our second environment variable will require an RPC URL. Some popular RPC connectivity services includeInfura ,LinkPool , andFiews . After establishing a valid RPC connection and the funded address, you can start deploying smart contracts to public networks.

*Shown above are the environment variables needed to link your MetaMask wallet to Truffle, found in truffle-config.js*Once you have all the data you need, you can set your environment variables via the terminal commands:

export

RPC_URL

your_url_hereexport

MNEMONIC

'your 12

words here'

Migration¶

We are now ready to deploy our smart contract on a public network. To begin, run the command:

npm run migrate:live This will compile all of your smart contracts, and then start deploying (migrating) them.

Note: you may encounter some compilation warnings with the Chainlink contracts. This is simply because the LINK token was originally deployed with an older version of Solidity. Given that this contract is only being deployed when you run tests, these can be ignored.

Console output of one of our contracts deploying. This particular transaction can be foundhere. Deploying ourMigrations andMyContract cost us Ethereum, hence why we needed the faucet from earlier. Now that our contract is deployed, we can utilize the three helper scripts to fundMyContract with LINK (which we got from the Chainlink faucet), create requests, and read the state.

There are 3 helper scripts provided to interact with our contract, located in the scripts directory:

- fund-contract.js
- request-data.js
- read-contract.js

They can be used by calling them fromnpx truffle exec . First, we must fund our contract with LINK, which can be done by running the following command:

npx truffle exec

scripts/fund-contract.js --network live

Following deployment, the output displays our contract address as well as the transaction address. We can also look this up on the ropstenexplorer .

Notice that upon completion, a payment of 1 LINK was sent during the transaction, clearly indicating that our contract has properly been funded. Next, we can nowrequest data from the contract by running:

npx truffle exec

scripts/request-data.js --network live *In the Chainlink explorer for the network that you made the request (in our example we use Ropsten, but it would work similarly for Rinkeby, Kovan, and Mainnet). As you can see the Chainlink node that we were requesting data from picked up our request and waited for 3 block confirmations before executing the job. We are now ready to read the contract's state with the written answer from the Chainlink node, indicated by the fulfillment transaction (highlighted in red).* And finally, for reading the data we run our read-contract script. In particular, this will run faster because we're not creating a transaction, we are simply reading the current state of the smart contract.

npx truffle exec

scripts/read-contract.js --network live

Notice for our output when reading the data, we get a number displaying the current price of LINK in USD * 100 (as of writing, 22094). This is due to the value of times specified in our smart contract request-data.js.

Request-data.js. TRUFFLE_CL_BOX_TIMES is an environment variable which is used to override the default value of 100. This value is used for the precision of the endpoint's response (in this case, for the price of ETH in USD), and also because Solidity cannot handle decimals. Any of these values can be changed for your project. Thanks to the power of Truffle, it has never been easier to work with smart contract development. We hope you enjoyed working with our Chainlink Truffle Box, allowing you to fully test, deploy, and interact with contracts on the network. Thanks for reading, and stay tuned for future updates athttps://blog.chain.link/ .