Contract Interaction

Introduction

Contract interactions , in blockchains networks, refers to the process of communicating with deployed smart contracts. This involves querying contract information or executing specific functions defined within the contract. Interactions can include tasks ranging from checking balances to transferring tokens, and so on.

Profiles provide network and contract configurations, while dynamic methods simplify common tasks. Scripts automate interactions for programmatic tasks. This interaction is vital for utilizing decentralized applications and services offered by smart contracts on the blockchain.

Start an interaction

You can interact with your project's contracts by using theshell orrun commands.

Getting started

You can reproduce the examples provided here, by initially adding and compiling a basicstarter ↗(opens in a new tab) contract and acw-20 token contract ↗(opens in a new tab)to your project with the following commands:

jenesis

add

contract

starter

my_first_contract

-d

deployment_1 jenesis

add

contract

cw20-base

my_token

-d

token_1 jenesis

compile For more information on contract template examples, visitJenesis Templates ↗(opens in a new tab)

Interactive shell

You can open a shell within which you can easily start an interaction with your contracts, by simply running:

jenesis

shell If a profile is not selected, the default profile will be selected automatically. You can specify any profile using the--profile optional argument:

jenesis

shell

--profile

my_profile You will observe the following text indicating the available contracts in your project:

Network: fetchai-testnet Detecting contracts... C deployment_1 C token_1 Detecting contracts...complete ℹ jenesis shell currently requires that contract names use accepted python variable names. For example, usingtoken-1 instead oftoken_1 will generate an error when trying to interact with it.** In this case, we can see thatdeployment_1 andtoken_1 deployments are available for this project. If these contracts have been already deployed, you can directly interact with them by

performing contract queries and executions, such as:

deployment_1 . query (args = { 'msg_name' : {...}}

deployment_1. execute (args = { 'msg_name' : {...}} A ledger client (ledger) and your locally stored wallet keys will also be available in the shell. For example, if you have a local key namedalice , you will find this underwallets['alice'] and you can query the balance as follows:

ledger . query_bank_balance (wallets['alice']) 10000000000000000000 If the ledger is a testnet with a faucet URL, you can get funds using thefaucet :

faucet . get_wealth (wallets['alice'])

Dynamic methods

Jenesis also attaches the contract query, execution and deploy messages as dynamic methods.

For instance, the following query:

token_1 . query ({ "balance" : { "address" : str (wallet. address ())}})) can also be run as follows:

token_1 . balance (address = str (wallet. address ())) { 'balance' :

'4000' } Similarly, instead of usingtoken_1.execute... , a transfer can be executed with the following:

token_1 . transfer (amount = '1000' , recipient = str (wallet2. address ()), sender = wallet) Additionally, Jenesis has an autocompletion helper for query, execution and deployment arguments. It will show automatically when typing in the shell.

We will now show an example assuming thattoken_1 deployment contract has only been compiled and not yet deployed, going throughdeployment ,execution , andquerying using dynamic methods.

For this example, we will first generate two wallets. We provide wealth to the sender wallet in atestfet so it can pay for transaction fees:

wallet

LocalWallet . generate ()

wallet2

LocalWallet . generate ()

faucet . get_wealth (wallet) We can then proceed to deploymy_token contract usingtoken_1 deployment configuration. We define the arguments for the CW-20 token:name ,symbol ,decimal , and theaddress that will be funded with these CW-20 tokens. In this case, we will fund wallet's address with 5000 tokens:

token_1 . deploy (name = "Crab Coin" , symbol = "CRAB" , decimals = 6 , initial_balances = [{ "address" : str (wallet. address ()), "amount" : "5000" }], sender = wallet) We canquery wallet balance to make sure it has been funded with cw20 tokens. Run the following:

token_1 . balance (address = str (wallet. address ())) { 'balance' :

'5000' } We can now execute a CW-20 tokentransfer of 1000 tokens from wallet to wallet2:

token_1 . transfer (amount = '1000' , recipient = str (wallet2. address ()), sender = wallet) Finally, wequery both wallets' balance :

token_1 . balance (address = str (wallet. address ())) { 'balance' :

'4000' }

token_1 . balance (address = str (wallet2. address ())) { 'balance' :

'1000' } We can observe thatwallet has sent 1000 tokens to wallet2 .

Executing scripts

You can also assemble the above commands into a script that is executable by therun command, in the following way:

from cosmpy . aerial . wallet import LocalWallet

wallet

LocalWallet . generate () faucet . get_wealth (wallet. address ()) wallet2 = LocalWallet . generate ()

token_1 . deploy (name = "Crab Coin" , symbol = "CRAB" , decimals = 6 , initial_balances = [{ "address" : str (wallet. address ()), "amount" : "5000" }], sender = wallet)

print ("wallet initial cw20 MT balance: " , token_1. balance (address = str (wallet. address ())))

tx

token_1 . transfer (amount = '1000' , recipient = str (wallet2. address ()), sender = wallet) print ("transfering 1000 cw20 MT tokens from wallet to wallet2") tx . wait_to_complete ()

print ("wallet final cw20 MT balance: " , token_1. balance (address = str (wallet. address ()))) print ("wallet2 final cw20 MT balance: " , token_1. balance (address = str (wallet2. address ()))) If you paste the above code into the filescript.py inside the project's directory, you can run it with:

jenesis run script.py And you will observe the same output as before. You can also specify the profile as an optional argument using--profile .

Finally, you can pass arguments to the script just as you would to a standard Python script by placing all the arguments to the script after the-- delimiter:

jenesis run script.py [--profile profile_name] -- arg1 arg2 --key1 value1 --key2 value2 For a better understanding, visit theCosmPy ↗ documentation for more contract interaction examples.

Was this page helpful?

How to deploy contracts Installation