Deployment and Interaction

We have the wasm binary ready. Now it is time to deploy it to the testnet and start interacting. You can use thewasmd Go CLI or theCosmJS Node Console as you prefer.

wasmd Go CLI

We generated a wasm binary executable in the previous chapter. Let's upload the code to the blockchain. Once that is complete, you can download the bytecode to verify it.

See the list of codes that was uploaded to the testnet previously.

wasmd query wasm list-code NODE

Here is an alternative if you haven't set up the environment variables to interact with the testnet previously:

wasmd query wasm list-code --node https://rpc.malaga-420.cosmwasm.com:443 Now let us store the bytecode on chain and acquire the Code Id. The Code Id will later be used to create an instance of the cw_namespace contract.

If you have already set up the environment variables, you can use the following command:

RES

(wasmd tx wasm store artifacts/cw_nameservice.wasm --from wallet TXFLAG -y --output json -b block)

Otherwise, you will have to type in the following command to upload the wasm binary to the testnet:

RES

(wasmd tx wasm store artifacts/cw_nameservice.wasm --from wallet --node https://rpc.malaga-420.cosmwasm.com:443 --chain-id malaga-420 --gas-prices 0 .25umlg --gas auto --gas-adjustment 1.3 -y --output json -b block)

The response contains the Code Id of the uploaded wasm binary.

echo

RES

The following is an easier way to get the Code Id from the response:

CODE_ID

(echo RES | jq -r '.logs[0].events[-1].attributes[0].value') echo

CODE_ID Let's see the list of contracts instantiated using the Code Id above.

wasmd query wasm list-contract-by-code CODE_ID

NODE --output json The response should be an empty list as we have not instantiated any contract yet.

{ "contracts" : [ ] , "pagination" : { "next_key" : null , "total" : "0" } } Before we instantiate a contract with the Code Id and interact with it, let us check if the code stored on the blockchain is indeed the cw_namespace.wasm binary we uploaded.

Download the wasm binary from the chain and compare it to the original one

wasmd query wasm code CODE_ID

NODE download.wasm

The two binaries should be identical

diff artifacts/cw_nameservice.wasm download.wasm

Instantiating the Contract

We can now create an instance of the wasm contract. Following the instantiation, we can make queries and this time receive non-empty responses.

Prepare the instantiation message

INIT

'{"purchase_price":{"amount":"100","denom":"umlg"},"transfer_price":{"amount":"999","denom":"umlg"}}'

Instantiate the contract

wasmd tx wasm instantiate CODE_ID

" INIT " --from wallet --label "name service"

TXFLAG -y --no-admin

Check the contract details and account balance

wasmd query wasm list-contract-by-code CODE_ID

NODE --output json CONTRACT = (wasmd query wasm list-contract-by-code CODE_ID NODE --output json | jq -r '.contracts[-1]') echo

CONTRACT

See the contract details

wasmd query wasm contract CONTRACT

NODE

Check the contract balance

wasmd query bank balances CONTRACT

NODE

Upon instantiation the cw_nameservice contract will store the instatiation message data in the contract's storage with the storage key "config".

Query the entire contract state

wasmd query wasm contract-state all CONTRACT

NODE models : - key : 0006636F6E666967 value : eyJwdXJjaGFzZV9wcmljZSI6eyJkZW5vbSI6InVtbGciLCJhbW91bnQiOiIxMDAifSwidHJhbnNmZXJfcHJpY2UiOnsiZGVub20iOiJ1bWxnIiwiYW1vdW50IjoiOTk5In19 pagination : next_key :

null total :

"0"

Note that the storage key "config" is hex encoded and prefixed with two bytes indicating its length

echo -n config | xxd -ps

gives 636f6e666967

thus we have the following output: 0006636f6e666967

You can also query a storage key directly

wasmd query wasm contract-state raw CONTRACT 0006636f6e666967 NODE --hex

The base64 encoded response is the stored value corresponding to the storage key "config" and reads as follows:

{"purchase_price":{"denom":"umlg","amount":"100"},"transfer_price":{"denom":"umlg","amount":"999"}}

Note that keys are hex encoded, and the values are base64 encoded.

To view the returned data (assuming it is ascii), try something like:

(Note that in many cases the binary data returned is not in ascii format, thus the encoding)

wasmd query wasm contract-state all CONTRACT NODE --output "json" | jq -r '.models[0].key' | xxd -r -ps wasmd query wasm contract-state all CONTRACT NODE --output "json" | jq -r '.models[0].value' | base64 -d

The cw_namespace contract implements a QueryMsg that returns the contents of the storage

So, we can also try "smart querying" the contract

CONFIG_QUERY='{"config": {}}' wasmd query wasm contract-state smart CONTRACT "CONFIG_QUERY" NODE --output json

and the contract will return the following response:

{"data":{"purchase_price": {"denom":"umlg","amount":"100"},"transfer_price": {"denom":"umlg","amount":"999"}}}

Contract Interaction

Now that the contract is instantiated, let's register a name and transfer it to another address by paying the transfer fee.

Register a name for the wallet address

REGISTER

'{"register":{"name":"fred"}}' wasmd tx wasm execute CONTRACT

" REGISTER " --amount 100umlg --from wallet TXFLAG -y

Query the owner of the name record

NAME_QUERY

'{"resolve_record": {"name": "fred"}}' wasmd query wasm contract-state smart CONTRACT

" NAME_QUERY "

NODE --output json

{"data":{"address":"wasm1pze5wsf0dg0fa4ysnttugn0m22ssf3t4a9yz3h"}}

Transfer the ownership of the name record to wallet2 (change the "to" address to wallet2 generated during environment setup)

wasmd keys list TRANSFER = '{"transfer":{"name":"fred","to":"wasm1lwcrr9ktsmn2f7fej6gywxcm8uvxlzz5ch40hm"}}' wasmd tx wasm execute CONTRACT

" TRANSFER " --amount 999umlg --from wallet TXFLAG -y

Query the record owner again to see the new owner address:

NAME_QUERY

'{"resolve_record": {"name": "fred"}}' wasmd query wasm contract-state smart CONTRACT

" NAME_QUERY "

NODE --output json

{"data":{"address":"wasm1lwcrr9ktsmn2f7fej6gywxcm8uvxlzz5ch40hm"}}

CosmJS Node Console

The binary can be uploaded to the chain, instantiated and interacted with using the CosmJS Node Console as well.

Open a new Terminal window and initialize a CosmJS CLI session with the following command:

npx @cosmjs/cli@^0.28.1 --init https://raw.githubusercontent.com/InterWasm/cw-plus-helpers/main/base.ts Now let us import the necessary utils, generate a wallet address and upload the wasm binary to the chain.

import

{ calculateFee ,

StdFee }

from

"@cosmjs/stargate" import

{ coin }

from

"@cosmjs/amino"

const

[addr , client]

=

await

useOptions (malagaOptions) . setup ("password") ; // Make sure you use the right path for the wasm binary const wasm = fs . readFileSync ("/home/username/.../cw-contracts/contracts/nameservice/artifacts/cw_nameservice.wasm") const uploadFee =

calculateFee (malagaOptions . fees . upload , malagaOptions . gasPrice) const result =

await client . upload (addr , wasm , uploadFee) console . log (result) { originalSize :

140819 , originalChecksum : 'f388fff922ffbb3845311608887636346eac6246ee20578e11e3577226466e7b' , compressedSize :

52176 , compressedChecksum : 'e5a1d70aedf1c2d6b1a3cbeec2981ac84c3ae48f642e221608b1624ccc6d65ba' , codeId :

17 , logs :

[

{ msg_index :

0 , log : '' , events :

[Array]

}

] , transactionHash : '34FA5BE9991578C56961C6BA7F5B25474BAFCE44954035217A93FE91BCC673A1' } Please note thecodeId of the uploaded wasm binary. This is the codeId that will be used to instantiate the contract.

You may compare the value oforiginalChecksum with the contents of the fileartifacts/checksums.txt to make sure the code on the chain is identical to thecw_nameservice.wasm binary.

For future reference, you may get the checksum for a given Code Id as follows:

//If you haven't created/loaded a wallet yet, you can do so with the following command: //const [addr, client] = await useOptions(malagaOptions).setup("password");

const codeIdToVerify =

17 const codes =

await client . getCodes ( ) const hash = codes . filter ((x)

=> x . id

=== codeIdToVerify) . map ((x)

=> x . checksum) [0] ; console . log (hash) You may then compile the contract code yourself, optimize the resulting binary and compare the checksum to the value ofhash to make sure the code on the chain was not tampered with.

Instantiating the Contract

Now, the wasm binary is uploaded to the chain with a Code Id and is ready to be instantiated.

First define adefaultFee to be passed into instantiation and execution functions later on:

const

defaultFee :

StdFee

=

{

amount :

[{ amount :

"200000" ,

denom :

"umlg" , } ,] ,

gas :

"200000" , } ; Create a cw_nameservice contract instance using the code id we have received upon upload.

//Replace the Code Id with the one you have received earlier const codeId =

17 //Define the instantiate message const instantiateMsg =

{ "purchase_price" : { "amount" : "100" , "denom" : "umlg" } , "transfer_price" : { "amount" : "999" , "denom" : "umlg" } } //Instantiate the contract const instantiateResponse =

await client . instantiate (addr , codeId , instantiateMsg ,

"Our Name Service" , defaultFee)

console . log (instantiateResponse) { contractAddress : 'wasm1s66zhks8v3fm24974crzxufh7w6ktt69jq8e3zt8q7cyvr52vlqqlzl7sc' , logs :

[

{ msg_index :

0 , log : '' , events :

[Array]

}

] , transactionHash : '27F4B6B41C36AB6B8324C125EFAE6C535093A6229CDF3665B998CD071E1566E2' } We've successfully instantiated the contract and have a contract address for the new instance.

The list of contracts instantiated using a certain codeId can be queried using the following command:

const contracts =

await client . getContracts (17) console . log (contracts) ;

Contract Interaction

Now let us register a name on the nameservice contract instance and transfer it to another address.

//Enable REPL editor mode to edit multiple lines of code . editor

const executeResponse =

await client . execute (addr , instantiateResponse . contractAddress , { register : { name : "fred" } } , defaultFee , '' , [coin (100 ,

"umlg")]) //Exit the editor using^D to execute the code entered ^ D Query the name record to see the owner address.

const queryResponse =

await client . queryContractSmart (instantiateResponse . contractAddress ,

{ resolve_record :

{ name : "fred" } }) console . log (queryResponse) //It should match your wallet address console . log (addr) Now, let us create another wallet and transfer the ownership of the name to the new wallet address.

const

[anotherAddress , anotherClient]

=

await

useOptions (malagaOptions) . setup ("password" , ".another_wallet.key") ;

//Enable REPL editor mode to edit multiple lines of code . editor

const executeResponse_2 =

await client . execute (addr , instantiateResponse . contractAddress , { transfer : { name : "fred" , to : anotherAddress } } , defaultFee , '' , [coin (999 ,

"umlg")]) Query the name record one more time to see the address for the new owner.

const queryResponse_2 =

await client . queryContractSmart (instantiateResponse . contractAddress ,

{ resolve_record :

{ name : "fred" } }) console . log (queryResponse_2) //The response should match the new wallet address after the change of ownership console . log (anotherAddress) Previous Downloading and Compiling a ContractNext Next Steps * wasmd Go CLI * * Instantiating the Contract * * Contract Interaction *