Deployment

On BandChain, an oracle script can be registered into the system by anyone. This is done through the registrant sending aMsgCreateOracleScript message to the chain.

AMsgCreateOracleScript message contains various parameters of the oracle script that is to be registered. These parameters include:

- name
- : Name of the oracle script.
- description
- : A description of the oracle script.
- schema
- : The oracle script's schema which details the inputs and outputs of this oracle script.
- source_code_url
- : The URL for the source code of the oracle script.
- code
- : The Owasm-compiled binary of the oracle script.
- owner
- : The owner of the oracle script. The owner will have edit rights. If omitted, the oracle script's parameters
- will no longer be able to be edited after being registered.
- sender
- : The message sender account.

In order to send aMsgCreateOracleScript message, we can use eithebandchain.js orpyband

An example on how to send aMsgCreateOracleScript message viabandchain.js can be seen below.

```
import

{

Client ,

Wallet ,

Message ,

Coin ,

Transaction ,

Fee

}

from

'@bandprotocol/bandchain.js' import

fs

from

'fs' import

path

from

'path'

const grpcURL =

'https://laozi-testnet6.bandchain.org/grpc-web' const client =

new

Client ( grpcURL )

// Setup the client async
```

```
function

createOracleScript ( )

{ // Setup the wallet const

{

PrivateKey

}

=

Wallet const mnemonic = process . env . MNEMONIC const privateKey =

PrivateKey . fromMnemonic ( mnemonic ) const publicKey = privateKey . toPubkey ( ) const sender = publicKey . toAddress ( ) . toAccBech32 ( )

// Setup the transaction's properties const chainId =

await client . getChainId ( ) const execPath = path . resolve ( __dirname ,

'hello_world.wasm' ) const code = fs . readFileSync ( execPath )

let feeCoin =

new

Coin ( ) feeCoin . setDenom ( 'uband' ) feeCoin . setAmount ( '0' )

const requestMessage =

new

Message . MsgCreateOracleScript ( 'Hello World!' ,

// oracle script name code ,

// oracle script code sender ,

// owner sender ,

// sender '' ,

// description '{repeat:u64}/{response:string}' ,

// schema 'https://ipfs.io/ipfs/QmSSrgJ6QuFDJHyC2SyTgnHKRBhPdLHUD2tJJ86xejrCfn'

// source code url )

// Construct the transaction const fee =

new

Fee ( ) fee . setAmountList ( [ feeCoin ] ) fee . setGasLimit ( 350000 )

const txn =

new

Transaction ( ) txn . withMessages ( requestMessage ) await txn . withSender ( client , sender ) txn . withChainId ( chainId ) txn . withFee ( fee ) txn . withMemo ( '' )

// Sign the transaction const signDoc = txn . getSignDoc ( publicKey ) const signature = privateKey . sign ( signDoc ) const txRawBytes = txn . getTxData ( signature , publicKey )

// Broadcast the transaction const sendTx =

await client . sendTxBlockMode ( txRawBytes )

return sendTx }

; ( async
```

( )

=>

{ console . log (await

createOracleScript ( )) }) ( ) An example on how to send aMsgCreateDataSource message via pyband can also be seen below.

import os

from pyband import Client , Transaction from pyband . wallet import PrivateKey from pyband . proto . cosmos . base . v1beta1 . coin_pb2 import Coin from pyband . proto . oracle . v1 . tx_pb2 import MsgCreateOracleScript from google . protobuf . json_format import MessageToJson

def

main ( ) :

Setup Client

grpc_url

"laozi-testnet6.bandchain.org" c = Client (grpc_url)

Setup Wallet

mnemonic

os . getenv ("MNEMONIC") private_key = PrivateKey . from_mnemonic (mnemonic) public_key = private_key . to_public_key ( ) sender_addr = public_key . to_address ( ) sender = sender_addr . to_acc_bech32 ( )

Prepare Transaction Properties

deploy_msg

MsgCreateOracleScript (name = "Hello World!" , description = "" , schema = "{repeat:u64}/{response:string}" , source_code_url = "https://ipfs.io/ipfs/QmSSrgJ6QuFDJHyC2SyTgnHKRBhPdLHUD2tJJ86xejrCfn" , code = open ("hello_world.wasm" ,

"rb") . read ( ) , owner = sender , sender = sender ,)

account

c . get_account (sender) account_num = account . account_number sequence = account . sequence

fee

[Coin (amount = "0" , denom = "uband")] chain_id = c . get_chain_id ( )

Construct a Transaction

txn

(Transaction ( ) . with_messages (deploy_msg) . with_sequence (sequence) . with_account_num (account_num) . with_chain_id (chain_id) . with_gas (250000) . with_fee (fee))

Sign the Transaction

sign_doc

txn . get_sign_doc (public_key) signature = private_key . sign (sign_doc . SerializeToString ( )) tx_raw_bytes = txn . get_tx_data (signature , public_key)

Broadcast the transaction

tx_block

c . send_tx_block_mode (bytes (tx_raw_bytes))

print (MessageToJson (tx_block))

if **name** ==

"**main**" : main ( ) After a successful transaction broadcast, the newly created oracle script ID can be found in the response json. The registrant can also view the created oracle script details onCosmoScan . An example of a successful transaction will return a response similar to the one shown below.

{ "height" :

"7440523" , "txhash" :

"FEDE0E7482CA6AB3A08E4643B2ADA03B0E6E961EE8747F41A1BF891BEDFE3C23" , "data" :

"0A220A202F6F7261636C652E76312E4D73674372656174654F7261636C65536372697074" , "rawLog" :

"[{\"events\":[{\"type\":\"create_oracle_script\",\"attributes\":[{\"key\":\"id\",\"value\":\"202\"}]},{\"type\":\"message\",\"attributes\":[{\"key\":\"action\",\"value\":\"/oracle.v1.MsgCreateOracleScript\"}]}]}]" , "logs" :

[{ "events" :

[{ "type" :

"create_oracle_script" , "attributes" :

[{ "key" :

"id" , "value" :

"202" }] } , { "type" :

"message" , "attributes" :

[{ "key" :

"action" , "value" :

"/oracle.v1.MsgCreateOracleScript" }] }] }] , "gasWanted" :

"250000" , "gasUsed" :

"246278" } Previous Creating an Oracle Script Next Supported Blockchains