Deployment

On BandChain, a data source can be registered into the system by anyone. This is done through the registrant sending aMsgCreateDataSource message to the chain.

AMsgCreateDataSource message contains various parameters of the data source that is to be registered. These parameters include:

- name
- : Name of the data source.
- description
- : A description of the data source .
- executable
- : Contents of the executable to be run by block validators upon receiving a data request for this data
- source.
- fee
- : A stipulated per-query fee that those who use this data source are required to pay.
- treasury
- : A treasury address of which the stipulated fee is to be paid to.
- owner
- : The owner address of this data source. The owner will have edit rights. If omitted, the data source
- parameters will no longer be able to be edited after being registered.
- sender
- : The message sender account.

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

An example on how to send aMsgCreateDataSource 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'

// Setup the client const grpcURL =

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

new

Client ( grpcURL )
```

async

function

createDataSource ( )

{ // 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.py') const file = fs . readFileSync (execPath ,

'utf8') const executable =

Buffer . from (file) . toString ('base64')

let feeCoin =

new

Coin ( ) feeCoin . setDenom ('uband') feeCoin . setAmount ('50000')

const requestMessage =

new

Message . MsgCreateDataSource ('Hello World!' ,

// Data source name executable ,

// Data source executable sender ,

// Treasury address sender ,

// Owner address sender ,

// Sender address [feeCoin] ,

// Fee ''

// Data source description)

// Construct the transaction const fee =

new

Fee ( ) fee . setAmountList ([feeCoin]) fee . setGasLimit (60000)

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

createDataSource ( )) }) ( ) 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 MsgCreateDataSource from google . protobuf . json_format import MessageToJson

def

main ( ) :

Setup the client

grpc_url

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

Setup the 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 the transaction's properties

deploy_msg

MsgCreateDataSource (name = "Hello World!" , description = "" , executable = open ("hello_world.py" ,

"rb") . read ( ) , fee = [Coin (amount = "0" , denom = "uband")] , treasury = sender , owner = sender , sender = sender ,)

account

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

fee

[Coin (amount = "50000" , denom = "uband")] chain_id = c . get_chain_id ( )

Construct the transaction

txn

(Transaction ( ) . with_messages (deploy_msg) . with_sequence (sequence) . with_account_num (account_num) . with_chain_id (chain_id) . with_gas (60000) . 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 the transaction is successfully broadcasted. The newly created data source ID can be found in the response json. The registrant can also view the created data source details onCosmoScan . An example of a successful transaction will return a response similar to the one shown below.

{ "height" :

"7232244" , "txhash" :

"B7A039E4AE9567A7380B5241B72FF69D38DB6831BE5023F05140616AD71FFA62" , "data" :

"0A200A1E2F6F7261636C652E76312E4D736743726561746544617461536F75726365" , "rawLog" :

"[{\"events\":[{\"type\":\"create_data_source\",\"attributes\":[{\"key\":\"id\",\"value\":\"327\"}]},{\"type\":\"message\",\"attributes\":[{\"key\":\"action\",\"value\":\"/oracle.v1.MsgCreateDataSource\"}]}]}]" , "logs" :

[{ "events" :

[{ "type" :

"create_data_source" , "attributes" :

[{ "key" :

"id" , "value" :

"327" }] } , { "type" :

"message" , "attributes" :

[{ "key" :

"action" , "value" :

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

"100000" , "gasUsed" :

"55306" } Previous Creating a Data Source Next Introduction