Wallet top-up

Introduction

The following guide is about how to top-up a wallet using the CosmPy library. To top-up a wallet, you need to create different wallets. In particular, if you are performing multiple transactions from a certaintask_wallet , you can set an algorithm to keep that wallet address topped-up. For this use case, we will use three different wallets:wallet ,authz_wallet , andtask_wallet .wallet will be the main wallet address that we don't want to give full access to, therefore we will authorizeauthz_wallet to send a certain amount of tokens fromwallet totask_wallet every timetask_wallet balance falls below a certainminimum_balance threshold. This way,task_wallet can keep performing transactions using the main wallet's tokens by being topped-up byauthz_wallet .

Walk-through

Aerial authorization: authorization address and authorization wallet

1. Let's start by creating a Python script for this:touch aerial_authz.py
2. First of all, we need to import the necessary classes:
3. import
4. argparse
5. from
6. datetime
7. import
8. datetime
9. ,
10. timedelta
11. from
12. google
13. .
14. protobuf
15. import
16. any_pb2
17. ,
18. timestamp_pb2
19. from
20. cosmpy
21. .
22. aerial
23. .
24. client
25. import
26. LedgerClient
27. ,
28. NetworkConfig
29. from
30. cosmpy
31. .
32. aerial
33. .
34. client
35. .
36. utils
37. import
38. prepare_and_broadcast_basic_transaction
39. from
40. cosmpy
41. .
42. aerial
43. .
44. faucet
45. import
46. FaucetApi
47. from
48. cosmpy
49. .
50. aerial

```
.
tx
import
Transaction
from
cosmpy
.
aerial
.
wallet
import
LocalWallet
from
cosmpy
.
protos
.
cosmos
.
authz
.
v1beta1
.
authz_pb2
import
Grant
from
cosmpy
.
protos
.
cosmos
.
authz
.
v1beta1
.
tx_pb2
import
MsgGrant
from
cosmpy
.
protos
.
cosmos
.
bank
.
v1beta1
.
authz_pb2
import
SendAuthorization
from
cosmpy
.
protos
.
cosmos
.
base
.
v1beta1
.
coin_pb2
import
Coin
```

We then proceed and define a_parse_commandline()
function:
def
_parse_commandline
():
parser
=
argparse
.
ArgumentParser
()
parser
.
add_argument
(
"authz_address"
,
help
=
"address that will be granted authorization to send tokens from wallet"
,
)
parser
.
add_argument
(
"total_authz_time"
,
type
=
int
,
nargs
=
"?"
,
default
=
10
,
help
=
"authorization time for authz_address in minutes"
,
)
parser
.
add_argument
(
"spend_limit"
,
type
=
int
,
nargs
=
"?"
,
default
=
1000000000000000000
,
help
=
"maximum tokens that authz_wallet will be able to spend from wallet"
,
)

187. return
188. parser
189. .
190. parse_args
191. ()
192. The_parse_commandline()
193. function is using theargparse
194. module to define a command-line interface for this script. It expects and processes three command-line arguments:
195. 1. authz_address
196. 1. : this is a required argument. It expects the user to provide an address that will be granted authorization to send tokens from a wallet.
197. 1. total_authz_time
198. 1. : this is an optional argument. If provided, it should be an integer representing the authorization time for theauthz_address
199. 1. in minutes. If not provided, it defaults to10
200. 1. minutes.
201. 1. spend_limit
202. 1. : this is another optional argument. If provided, it should be an integer representing the maximum tokens that theauthz_wallet
203. 1. will be able to spend from the wallet. If not provided, it defaults to10000000000000000000
204. 1. .
205. Thehelp
206. parameter provides a description of what each argument is for, which can be helpful for users who might not be familiar with the script. After defining these arguments, the function usesparser.parse_args()
207. to process the command-line arguments provided by the user and return them as an object containing the values provided forauthz_address
208. ,total_authz_time
209. , andspend_limit
210. .
211. We can the define ourmain()
212. function:
213. def
214. main
215. ():
216. """Run main."""
217. args
218. =
219. _parse_commandline
220. ()
221. wallet
222. =
223. LocalWallet
224. .
225. generate
226. ()
227. authz_address
228. =
229. args
230. .
231. authz_address
232. ledger
233. =
234. LedgerClient
235. (NetworkConfig.
236. fetchai_stable_testnet
237. ())
238. faucet_api
239. =

FaucetApi
(NetworkConfig.
fetchai_stable_testnet
())
total_authz_time
=
args
.
total_authz_time
wallet_balance
=
ledger
.
query_bank_balance
(wallet.
address
())
amount
=
args
.
spend_limit
while
wallet_balance
<
(amount)
:
print
(
"Providing wealth to wallet..."
)
faucet_api
.
get_wealth
(wallet.
address
())
wallet_balance
=
ledger
.
query_bank_balance
(wallet.
address
())
spend_amount
=
Coin
(amount
=
str
(amount), denom
=
"atestfet"
)

Authorize authz_wallet to send tokens from wallet

authz_any
=
any_pb2
.
Any
()
authz_any
.
Pack

```
(
SendAuthorization
(spend_limit
=
[spend_amount]),
""
,
)
expiry
=
timestamp_pb2
.
Timestamp
()
expiry
.
FromDatetime
(datetime.
now
()
+
timedelta
(seconds
=
total_authz_time
*
60
))
grant
=
Grant
(authorization
=
authz_any, expiration
=
expiry)
msg
=
MsgGrant
(
granter
=
str
(wallet.
address
()),
grantee
=
authz_address,
grant
=
grant,
)
tx
=
Transaction
()
tx
.
add_message
(msg)
tx
=
prepare_and_broadcast_basic_transaction
(ledger, tx, wallet)
tx
.
wait_to_complete
```

373. ()
374. if
375. **name**
376. ==
377. "**main**"
378. :
379. main
380. ()
381. In the first line we define a variableargs
382. using_parse_commandline()
383. function defined previously to retrieve the command-line argumentsauthz_address
384. ,total_authz_time
385. , andspend_limit
386. . The values are stored in theargs
387. variable. We then generate a new wallet using thegenerate()
388. method of theLocalWallet
389. class, and then set theauthz_address
390. variable to retrieve theauthz_address
391. from the command-line arguments previously defined. This is the address that will be granted authorization to send tokens from the wallet. We then initialize aledger
392. object using theLedgerClient
393. class and configure it to connect to the Fetch.ai stable testnet. We also initialize afaucet_api
394. object using theFaucetApi
395. class to provide access to a faucet API on the Fetch.ai stable testnet.
396. total_authz_time
397. retrieves the total authorization time (in minutes) from the command-line arguments. We proceed by checking the balance of the wallet by querying the ledger, using thequery_bank_balance()
398. method. We can then define a loop that continues until the wallet balance is greater than the specified spend amount (amount
399.). Within the loop, it requests additional tokens from the faucet usingfaucet_api.get_wealth()
400. and updates the wallet balance.
401. Below, we define thespend_amount
402. variable using a Coin object representing the spend amount. In this case, it's specified in the "atestfet" denomination. We then constructs an authorization object (authz_any
403.) usingSendAuthorization
404. . It sets an expiration time for the authorization, and creates an instance ofMsgGrant
405. message type, specifying thegranter
406. (the wallet's address),grantee
407. (theauthz_address
408.), and thegrant
409. (the authorization object). A new transaction (tx
410.) is finally created, andmsg
411. is added to it. The transaction is then prepared and broadcasted usingprepare_and_broadcast_basic_transaction()
412. . Finally, the script waits for the transaction to complete.
413. Save the script.

The overall script should be as follows.

aerial_authz.py import argparse from datetime import datetime , timedelta

from google . protobuf import any_pb2 , timestamp_pb2

from cosmpy . aerial . client import LedgerClient , NetworkConfig from cosmpy . aerial . client . utils import prepare_and_broadcast_basic_transaction from cosmpy . aerial . faucet import FaucetApi from cosmpy . aerial . tx import Transaction from cosmpy . aerial . wallet import LocalWallet from cosmpy . protos . cosmos . authz . v1beta1 . authz_pb2 import Grant from cosmpy . protos . cosmos . authz . v1beta1 . tx_pb2 import MsgGrant from cosmpy . protos . cosmos . bank . v1beta1 . authz_pb2 import SendAuthorization from cosmpy . protos . cosmos . base . v1beta1 . coin_pb2 import Coin

def

_parse_commandline (): parser = argparse . ArgumentParser () parser . add_argument ("authz_address" , help = "address that will be granted authorization to send tokens from wallet" ,) parser . add_argument ("total_authz_time" , type = int , nargs = "?" , default = 10 , help = "authorization time for authz_address in minutes" ,) parser . add_argument ("spend_limit" , type = int , nargs = "?" , default = 1000000000000000000 , help = "maximum tokens that authz_wallet will be able to spend from wallet" ,)

return parser . parse_args ()

def

main (): """Run main."""" args =

_parse_commandline ()

wallet

LocalWallet . generate ()

authz_address

args . authz_address

ledger

LedgerClient (NetworkConfig. fetchai_stable_testnet ()) faucet_api =

FaucetApi (NetworkConfig. fetchai_stable_testnet ())

total_authz_time

args . total_authz_time wallet_balance = ledger . query_bank_balance (wallet. address ())

amount

args . spend_limit

while wallet_balance < (amount) : print ("Providing wealth to wallet...") faucet_api . get_wealth (wallet. address ()) wallet_balance = ledger . query_bank_balance (wallet. address ())

spend_amount

Coin (amount = str (amount), denom = "atestfet")

Authorize authz_wallet to send tokens from wallet

authz_any

any_pb2 . Any () authz_any . Pack (SendAuthorization (spend_limit = [spend_amount]), "" ,)

expiry

timestamp_pb2 . Timestamp () expiry . FromDatetime (datetime. now () +

timedelta (seconds = total_authz_time *

60)) grant =

Grant (authorization = authz_any, expiration = expiry)

msg

MsgGrant (granter = str (wallet. address ()), grantee = authz_address, grant = grant,)

tx

Transaction () tx . add_message (msg)

tx

prepare_and_broadcast_basic_transaction (ledger, tx, wallet) tx . wait_to_complete ()

if

name

==

"**main**" : main ()

Aerial top-up

We are now ready to write a Python script which automates the process of topping-up the designated wallet (task_wallet) from the main wallet (wallet) after authorization fromauthz_wallet , whenever the balance oftask_wallet falls below a certain threshold (minimum_balance).

1. Let's create a Python script for this and name it:touch aerial_topup.py
2. Let's then import the needed modules such asargparse
3. for command-line argument parsing and modules from thecosmpy
4. library for blockchain interaction:
5. import
6. argparse
7. import
8. time
9. from
10. google
11. .
12. protobuf
13. import
14. any_pb2
15. from
16. cosmpy
17. .
18. aerial
19. .
20. client
21. import
22. LedgerClient
23. ,
24. NetworkConfig
25. from
26. cosmpy
27. .
28. aerial
29. .
30. client
31. .
32. utils
33. import
34. prepare_and_broadcast_basic_transaction
35. from
36. cosmpy
37. .
38. aerial
39. .
40. faucet
41. import
42. FaucetApi
43. from
44. cosmpy
45. .
46. aerial
47. .
48. tx
49. import
50. Transaction

from
cosmpy
.
aerial
.
wallet
import
LocalWallet
from
cosmpy
.
protos
.
cosmos
.
authz
.
v1beta1
.
tx_pb2
import
MsgExec
from
cosmpy
.
protos
.
cosmos
.
bank
.
v1beta1
.
tx_pb2
import
MsgSend
from
cosmpy
.
protos
.
cosmos
.
base
.
v1beta1
.
coin_pb2
import
Coin
We then define a_parse_commandline()
function that sets up command-line arguments:
def
_parse_commandline
():
parser
=
argparse
.
ArgumentParser
()
parser
.
add_argument
(
"wallet_address"
, help
=

```
"main wallet address"
)
parser
.
add_argument
(
"task_wallet_address"
, help
=
"wallet address that will perform transactions"
)
parser
.
add_argument
(
"top_up_amount"
,
type
=
int
,
nargs
=
"?"
,
default
=
10000000000000000
,
help
=
"top-up amount from wallet address to task_wallet address"
,
)
parser
.
add_argument
(
"minimum_balance"
,
type
=
int
,
nargs
=
"?"
,
default
=
1000000000000000
,
help
=
"minimum task_wallet address balance that will trigger top-up"
,
)
parser
.
add_argument
(
"interval_time"
,
type
=
int
,
nargs
```

=
"?"
,
default
=
5
,
help
=
"interval time in seconds to query task_wallet's balance"
,
)
return
parser
.
parse_args
()
Above we defined different arguments including the addresses of the main wallet (wallet_address
) and the task wallet (task_wallet_address
), the top-up amount fromwallet_address
totask_wallet_address
(top_up_amount
), the minimum balance fortask_wallet_address
(minimum_balance
), and the interval time in seconds to querytask_wallet_address
's balance (interval_time
).
After these arguments are defined, the function usesparser.parse_args()
to process the command-line arguments provided by the user. The values are then returned as an object, where each attribute corresponds to the name of the argument. This allows the script to access and utilize these values during execution.
We are now ready to define ourmain()
function:
def
main
():
"""Run main."""
ledger
=
LedgerClient
(NetworkConfig.
fetchai_stable_testnet
())
args
=
_parse_commandline
()
wallet_address
=
args
.
wallet_address
task_wallet_address
=
args
.
task_wallet_address

Use aerial_authz.py to authorize authz_wallet address to send tokens from wallet

authz_wallet
=
LocalWallet
.
generate

```
()
faucet_api
=
FaucetApi
(NetworkConfig.
fetchai_stable_testnet
())
wallet_balance
=
ledger
.
query_bank_balance
(authz_wallet.
address
())
while
wallet_balance
<
(
10
**
18
)
:
print
(
"Providing wealth to wallet..."
)
faucet_api
.
get_wealth
(authz_wallet.
address
())
wallet_balance
=
ledger
.
query_bank_balance
(authz_wallet.
address
())
ledger
=
LedgerClient
(NetworkConfig.
latest_stable_testnet
())
```

Top-up amount

```
amount
=
args
.
top_up_amount
top_up_amount
=
Coin
(amount
=
str
(amount), denom
=
"atestfet"
)
```

Minimum balance for task_wallet

```
minimum_balance
=
args
.
minimum_balance
```

Interval to query task_wallet's balance

```
interval_time
=
args
.
interval_time
while
True
:
wallet_balance
=
ledger
.
query_bank_balance
(wallet_address)
if
wallet_balance
<
amount
:
print
(
"Wallet doesn't have enough balance to top-up task_wallet"
)
break
task_wallet_balance
=
ledger
.
query_bank_balance
(task_wallet_address)
if
task_wallet_balance
<
minimum_balance
:
print
(
"topping up task wallet"
)
```

Top-up task_wallet

```
msg
=
any_pb2
.
Any
()
msg
.
Pack
(
MsgSend
(
```

from_address
=
wallet_address,
to_address
=
task_wallet_address,
amount
=
[top_up_amount],
),
""
,
)
tx
=
Transaction
()
tx
.
add_message
(
MsgExec
(grantee
=
str
(authz_wallet.
address
()), msgs
=
[msg]))
tx
=
prepare_and_broadcast_basic_transaction
(ledger, tx, authz_wallet)
tx
.
wait_to_complete
()
time
.
sleep
(interval_time)
if
name
==
"**main**"
:
main
()
Here we defined themain()
function which orchestrates all of the operations. It first initializes aledger
object to interact with the blockchain using theLedgerClient()
class. It then parses command-line arguments using_parse_commandline()
and stores them in theargs
variable. The function then retrieves wallet addresses forwallet
andtask_wallet
fromargs
. The function then usesaerial_authz.py
script previously created above to authorizeauthz_wallet
address to send tokens fromwallet
. If the balance ofauthz_wallet
is below10**18
, it uses a faucet API to provide wealth to the wallet until it reaches this threshold. Within the script, we then re-initialize the ledger object with the latest stable testnet configuration. We then proceed to set the top-up amount, the minimum balance, and interval timer thresholds fromargs
. The script then enters an infinite loop (while True
) in which it queries the balance of the mainwallet
. Checks if the main wallet has enough balance to top-uptask_wallet

437. . Queries the balance oftask_wallet
438. : if its balance falls below the specified minimum, it initiates a top-up by first creating a message to send tokens fromwallet_address
439. totask_wallet_address
440. , then constructing a transaction (tx
441.) with the authorization and message. It then prepares, broadcasts the transaction, and waits for a specified interval before repeating the process.
442. Save the script.

The overall script should be as follows:

aerial_topup.py import argparse import time

from google . protobuf import any_pb2

from cosmpy . aerial . client import LedgerClient , NetworkConfig from cosmpy . aerial . client . utils import prepare_and_broadcast_basic_transaction from cosmpy . aerial . faucet import FaucetApi from cosmpy . aerial . tx import Transaction from cosmpy . aerial . wallet import LocalWallet from cosmpy . protos . cosmos . authz . v1beta1 . tx_pb2 import MsgExec from cosmpy . protos . cosmos . bank . v1beta1 . tx_pb2 import MsgSend from cosmpy . protos . cosmos . base . v1beta1 . coin_pb2 import Coin

def

_parse_commandline (): parser = argparse . ArgumentParser () parser . add_argument ("wallet_address" , help = "main wallet address") parser . add_argument ("task_wallet_address" , help = "wallet address that will perform transactions") parser . add_argument ("top_up_amount" , type = int , nargs = "?" , default = 10000000000000000 , help = "top-up amount from wallet address to task_wallet address" ,) parser . add_argument ("minimum_balance" , type = int , nargs = "?" , default = 1000000000000000 , help = "minimum task_wallet address balance that will trigger top-up" ,) parser . add_argument ("interval_time" , type = int , nargs = "?" , default = 5 , help = "interval time in seconds to query task_wallet's balance" ,)

return parser . parse_args ()

def

main (): """Run main."""" ledger =

LedgerClient (NetworkConfig. fetchai_stable_testnet ()) args =

_parse_commandline ()

wallet_address

args . wallet_address

task_wallet_address

args . task_wallet_address

Use aerial_authz.py to authorize authz_wallet address to send tokens from wallet

authz_wallet

LocalWallet . generate () faucet_api =

FaucetApi (NetworkConfig. fetchai_stable_testnet ())

wallet_balance

ledger . query_bank_balance (authz_wallet. address ())

while wallet_balance < (10 ** 18) : print ("Providing wealth to wallet...") faucet_api . get_wealth (authz_wallet. address ()) wallet_balance = ledger . query_bank_balance (authz_wallet. address ())

ledger

LedgerClient (NetworkConfig. latest_stable_testnet ())

Top-up amount

amount

args . top_up_amount top_up_amount =

Coin (amount = str (amount), denom = "atestfet")

Minimum balance for task_wallet

minimum_balance

args . minimum_balance

Interval to query task_wallet's balance

interval_time

args . interval_time

while

True :

wallet_balance

ledger . query_bank_balance (wallet_address)

if wallet_balance < amount : print ("Wallet doesn't have enough balance to top-up task_wallet") break

task_wallet_balance

ledger . query_bank_balance (task_wallet_address)

if task_wallet_balance < minimum_balance :

print ("topping up task wallet")

Top-up task_wallet

msg

any_pb2 . Any () msg . Pack (MsgSend (from_address = wallet_address, to_address = task_wallet_address, amount = [top_up_amount],), "" ,)

tx

Transaction () tx . add_message (MsgExec (grantee = str (authz_wallet. address ()), msgs = [msg]))

tx

prepare_and_broadcast_basic_transaction (ledger, tx, authz_wallet) tx . wait_to_complete ()

time . sleep (interval_time)

if

name

==

"**main**" : main ()

Was this page helpful?

Oracles _ Liquidity pool