

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 certain task_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, and task_wallet. wallet will be the main wallet address that we don't want to give full access to, therefore we will authorize authz_wallet to send a certain amount of tokens from wallet to task_wallet every time task_wallet balance falls below a certain minimum_balance threshold. This way, task_wallet can keep performing transactions using the main wallet's tokens by being topped-up by authz_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

```
51. .
52. tx
53. import
54. Transaction
55. from
56. cosmpy
57. .
58. aerial
59. .
60. wallet
61. import
62. LocalWallet
63. from
64. cosmpy
65. .
66. protos
67. .
68. cosmos
69. .
70. authz
71. .
72. v1beta1
73. .
74. authz_pb2
75. import
76. Grant
77. from
78. cosmpy
79. .
80. protos
81. .
82. cosmos
83. .
84. authz
85. .
86. v1beta1
87. .
88. tx_pb2
89. import
90. MsgGrant
91. from
92. cosmpy
93. .
94. protos
95. .
96. cosmos
97. .
98. bank
99. .
100. v1beta1
101. .
102. authz_pb2
103. import
104. SendAuthorization
105. from
106. cosmpy
107. .
108. protos
109. .
110. cosmos
111. .
112. base
113. .
114. v1beta1
115. .
116. coin_pb2
117. import
118. Coin
```

```
119. We then proceed and define a_parse_commandline()
120. function:
121. def
122. _parse_commandline
123. ():
124. parser
125. =
126. argparse
127. .
128. ArgumentParser
129. ()
130. parser
131. .
132. add_argument
133. (
134. "authz_address"
135. ,
136. help
137. =
138. "address that will be granted authorization to send tokens from wallet"
139. ,
140. )
141. parser
142. .
143. add_argument
144. (
145. "total_authz_time"
146. ,
147. type
148. =
149. int
150. ,
151. nargs
152. =
153. "?"
154. ,
155. default
156. =
157. 10
158. ,
159. help
160. =
161. "authorization time for authz_address in minutes"
162. ,
163. )
164. parser
165. .
166. add_argument
167. (
168. "spend_limit"
169. ,
170. type
171. =
172. int
173. ,
174. nargs
175. =
176. "?"
177. ,
178. default
179. =
180. 1000000000000000000
181. ,
182. help
183. =
184. "maximum tokens that authz_wallet will be able to spend from wallet"
185. ,
186. )
```

```

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 to1000000000000000000
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. =

```

```

240. FaucetApi
241. (NetworkConfig.
242. fetchai_stable_testnet
243. ())
244. total_authz_time
245. =
246. args
247. .
248. total_authz_time
249. wallet_balance
250. =
251. ledger
252. .
253. query_bank_balance
254. (wallet.
255. address
256. ())
257. amount
258. =
259. args
260. .
261. spend_limit
262. while
263. wallet_balance
264. <
265. (amount)
266. :
267. print
268. (
269. "Providing wealth to wallet..."
270. )
271. faucet_api
272. .
273. get_wealth
274. (wallet.
275. address
276. ())
277. wallet_balance
278. =
279. ledger
280. .
281. query_bank_balance
282. (wallet.
283. address
284. ())
285. spend_amount
286. =
287. Coin
288. (amount
289. =
290. str
291. (amount), denom
292. =
293. "atestfet"
294. )

```

295. **Authorize authz_wallet to send tokens from wallet**

```

296. authz_any
297. =
298. any_pb2
299. .
300. Any
301. ()
302. authz_any
303. .
304. Pack

```

```
305. (  
306. SendAuthorization  
307. (spend_limit  
308. =  
309. [spend_amount]),  
310. ""  
311. ,  
312. )  
313. expiry  
314. =  
315. timestamp_pb2  
316. .  
317. Timestamp  
318. ()  
319. expiry  
320. .  
321. FromDatetime  
322. (datetime.  
323. now  
324. ()  
325. +  
326. timedelta  
327. (seconds  
328. =  
329. total_authz_time  
330. *  
331. 60  
332. ))  
333. grant  
334. =  
335. Grant  
336. (authorization  
337. =  
338. authz_any, expiration  
339. =  
340. expiry)  
341. msg  
342. =  
343. MsgGrant  
344. (  
345. granter  
346. =  
347. str  
348. (wallet.  
349. address  
350. ()),  
351. grantee  
352. =  
353. authz_address,  
354. grant  
355. =  
356. grant,  
357. )  
358. tx  
359. =  
360. Transaction  
361. ()  
362. tx  
363. .  
364. add_message  
365. (msg)  
366. tx  
367. =  
368. prepare_and_broadcast_basic_transaction  
369. (ledger, tx, wallet)  
370. tx  
371. .  
372. wait_to_complete
```

```

373. ()
374. if
375. name
376. ==
377. "main"
378. :
379. main
380. ()
381. In the first line we define a variable args
382. using _parse_commandline()
383. function defined previously to retrieve the command-line arguments authz_address
384. ,total_authz_time
385. , andspend_limit
386. . The values are stored in the args
387. variable. We then generate a new wallet using generate()
388. method of the LocalWallet
389. class, and then set the authz_address
390. variable to retrieve the authz_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 a ledger
392. object using the LedgerClient
393. class and configure it to connect to the Fetch.ai stable testnet. We also initialize a faucet_api
394. object using the FaucetApi
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 the query_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 using faucet_api.get_wealth()
400. and updates the wallet balance.
401. Below, we define the spend_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. ) using SendAuthorization
404. . It sets an expiration time for the authorization, and creates an instance of MsgGrant
405. message type, specifying the granter
406. (the wallet's address), grantee
407. (the authz_address
408. ), and the grant
409. (the authorization object). A new transaction (tx
410. ) is finally created, and msg
411. is added to it. The transaction is then prepared and broadcasted using prepare_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 from authz_wallet , whenever the balance of task_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 as argparse
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

```
51. from
52. cosmpy
53. .
54. aerial
55. .
56. wallet
57. import
58. LocalWallet
59. from
60. cosmpy
61. .
62. protos
63. .
64. cosmos
65. .
66. authz
67. .
68. v1beta1
69. .
70. tx_pb2
71. import
72. MsgExec
73. from
74. cosmpy
75. .
76. protos
77. .
78. cosmos
79. .
80. bank
81. .
82. v1beta1
83. .
84. tx_pb2
85. import
86. MsgSend
87. from
88. cosmpy
89. .
90. protos
91. .
92. cosmos
93. .
94. base
95. .
96. v1beta1
97. .
98. coin_pb2
99. import
100. Coin
101. We then define a _parse_commandline()
102. function that sets up command-line arguments:
103. def
104. _parse_commandline
105. ():
106. parser
107. =
108. argparse
109. .
110. ArgumentParser
111. ()
112. parser
113. .
114. add_argument
115. (
116. "wallet_address"
117. , help
118. =
```

```
119. "main wallet address"
120. )
121. parser
122. .
123. add_argument
124. (
125. "task_wallet_address"
126. , help
127. =
128. "wallet address that will perform transactions"
129. )
130. parser
131. .
132. add_argument
133. (
134. "top_up_amount"
135. ,
136. type
137. =
138. int
139. ,
140. nargs
141. =
142. "?"
143. ,
144. default
145. =
146. 1000000000000000000
147. ,
148. help
149. =
150. "top-up amount from wallet address to task_wallet address"
151. ,
152. )
153. parser
154. .
155. add_argument
156. (
157. "minimum_balance"
158. ,
159. type
160. =
161. int
162. ,
163. nargs
164. =
165. "?"
166. ,
167. default
168. =
169. 1000000000000000000
170. ,
171. help
172. =
173. "minimum task_wallet address balance that will trigger top-up"
174. ,
175. )
176. parser
177. .
178. add_argument
179. (
180. "interval_time"
181. ,
182. type
183. =
184. int
185. ,
186. nargs
```

```

187. =
188. "?"
189. ,
190. default
191. =
192. 5
193. ,
194. help
195. =
196. "interval time in seconds to query task_wallet's balance"
197. ,
198. )
199. return
200. parser
201. .
202. parse_args
203. ()
204. Above we defined different arguments including the addresses of the main wallet (wallet_address
205. ) and the task wallet (task_wallet_address
206. ), the top-up amount fromwallet_address
207. totask_wallet_address
208. (top_up_amount
209. ), the minimum balance fortask_wallet_address
210. (minimum_balance
211. ), and the interval time in seconds to querytask_wallet_address
212. 's balance (interval_time
213. ).
214. After these arguments are defined, the function usesparser.parse_args()
215. 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.
216. We are now ready to define ourmain()
217. function:
218. def
219. main
220. ():
221. """Run main."""
222. ledger
223. =
224. LedgerClient
225. (NetworkConfig.
226. fetchai_stable_testnet
227. ())
228. args
229. =
230. _parse_commandline
231. ()
232. wallet_address
233. =
234. args
235. .
236. wallet_address
237. task_wallet_address
238. =
239. args
240. .
241. task_wallet_address

```

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

```

243. authz_wallet
244. =
245. LocalWallet
246. .
247. generate

```

```

248. ()
249. faucet_api
250. =
251. FaucetApi
252. (NetworkConfig.
253. fetchai_stable_testnet
254. ())
255. wallet_balance
256. =
257. ledger
258. .
259. query_bank_balance
260. (authz_wallet.
261. address
262. ())
263. while
264. wallet_balance
265. <
266. (
267. 10
268. **
269. 18
270. )
271. :
272. print
273. (
274. "Providing wealth to wallet..."
275. )
276. faucet_api
277. .
278. get_wealth
279. (authz_wallet.
280. address
281. ())
282. wallet_balance
283. =
284. ledger
285. .
286. query_bank_balance
287. (authz_wallet.
288. address
289. ())
290. ledger
291. =
292. LedgerClient
293. (NetworkConfig.
294. latest_stable_testnet
295. ())

```

296. **Top-up amount**

```

297. amount
298. =
299. args
300. .
301. top_up_amount
302. top_up_amount
303. =
304. Coin
305. (amount
306. =
307. str
308. (amount), denom
309. =
310. "atestfet"
311. )

```

312. **Minimum balance for task_wallet**

```
313. minimum_balance
314. =
315. args
316. .
317. minimum_balance
```

318. **Interval to query task_wallet's balance**

```
319. interval_time
320. =
321. args
322. .
323. interval_time
324. while
325. True
326. :
327. wallet_balance
328. =
329. ledger
330. .
331. query_bank_balance
332. (wallet_address)
333. if
334. wallet_balance
335. <
336. amount
337. :
338. print
339. (
340. "Wallet doesn't have enough balance to top-up task_wallet"
341. )
342. break
343. task_wallet_balance
344. =
345. ledger
346. .
347. query_bank_balance
348. (task_wallet_address)
349. if
350. task_wallet_balance
351. <
352. minimum_balance
353. :
354. print
355. (
356. "topping up task wallet"
357. )
```

358. **Top-up task_wallet**

```
359. msg
360. =
361. any_pb2
362. .
363. Any
364. ()
365. msg
366. .
367. Pack
368. (
369. MsgSend
370. (
```

```

371. from_address
372. =
373. wallet_address,
374. to_address
375. =
376. task_wallet_address,
377. amount
378. =
379. [top_up_amount],
380. ),
381. ""
382. ,
383. )
384. tx
385. =
386. Transaction
387. ()
388. tx
389. .
390. add_message
391. (
392. MsgExec
393. (grantee
394. =
395. str
396. (authz_wallet.
397. address
398. ()), msgs
399. =
400. [msg]))
401. tx
402. =
403. prepare_and_broadcast_basic_transaction
404. (ledger, tx, authz_wallet)
405. tx
406. .
407. wait_to_complete
408. ()
409. time
410. .
411. sleep
412. (interval_time)
413. if
414. name
415. ==
416. "main"
417. :
418. main
419. ()
420. Here we defined themain()
421. function which orchestrates all of the operations. It first initializes a ledger
422. object to interact with the blockchain using theLedgerClient()
423. class. It then parses command-line arguments using _parse_commandline()
424. and stores them in theargs
425. variable. The function then retrieves wallet addresses forwallet
426. andtask_wallet
427. fromargs
428. . The function then usesaerial_authz.py
429. script previously created above to authorizeauthz_wallet
430. address to send tokens fromwallet
431. . If the balance ofauthz_wallet
432. is below10**18
433. , 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
434. . The script then enters an infinite loop (while True
435. ) in which it queries the balance of the mainwallet
436. . Checks if the main wallet has enough balance to top-upwallet

```

```

437. . Queries the balance of task_wallet
438. : if its balance falls below the specified minimum, it initiates a top-up by first creating a message to send tokens
    from wallet_address
439. to task_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
= 10000000000000000 , 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](#)