Liquidity pool

Introduction

This Liquidity pool interaction guide provides a practical demonstration of interacting with a liquidity pool on the Fetch.ai network. This script showcases various operations, including swapping assets, providing liquidity, and withdrawing liquidity, utilizing smart contracts and local wallets.

Walk-through

1. Let's start by creating a Python script for this and name it:touch aerial_liquidity_pool.py
2. Let's then import the needed modules:
3. import
4. argparse
5. import
6. base64
7. from
8. cosmpy
9. .
10. aerial
11. .
12. client
13. import
14. LedgerClient
15. ,
16. NetworkConfig
17. from
18. cosmpy
19. .
20. aerial
21. .
22. contract
23. import
24. LedgerContract
25. from
26. cosmpy
27. .
28. aerial
29. .
30. faucet
31. import
32. FaucetApi
33. from
34. cosmpy
35. .
36. aerial
37. .
38. wallet
39. import
40. LocalWallet
41. We need to define a_parse_commandline()
42. function:
43. def
44. _parse_commandline
45. ():
46. parser
47. =
48. argparse
49. .
50. ArgumentParser
51. ()
52. parser
53. .
54. add_argument
55. (
56. "swap_amount"

```
,
type
=
int
,
nargs
=
"?"
,
default
=
10000
,
help
=
"atestfet swap amount to get some cw20 tokens on wallet's address"
,
)
parser
.
add_argument
(
"cw20_liquidity_amount"
,
type
=
int
,
nargs
=
"?"
,
default
=
100
,
help
=
"amount of cw20 tokens that will be provided to LP"
,
)
parser
.
add_argument
(
"native_liquidity_amount"
,
type
=
int
,
nargs
=
"?"
,
default
=
2470
,
help
=
"amount of atestfet tokens that will be provided to LP"
,
)
return
parser
.
parse_args
```

125. ()
126. The function expects and processes three command-line arguments:
127.
    - swap_amount
128.
    - : this argument specifies the amount of atestfet tokens to swap in order to receive some cw20 tokens on the wallet's address. It is an optional argument, and if not provided, it defaults to10000
129.
    - .
130.
    - cw20_liquidity_amount
131.
    - : this argument sets the amount of cw20 tokens that will be provided to the liquidity pool. It is also optional and defaults to100
132.
    - if not provided.
133.
    - native_liquidity_amount
134.
    - : this argument represents the amount of atestfet tokens that will be provided to the liquidity pool. Like the others, it is optional and defaults to2470
135.
    - if not specified.
136. Thehelp
137. parameter for each argument provides a description of what it is used for. The function then usesparser.parse_args()
138. to process the command-line arguments provided by the user and return them as an object containing the values forswap_amount
139. ,cw20_liquidity_amount
140. , andnative_liquidity_amount
141. .
142. We are now ready to define ourmain()
143. function, which orchestrates the interaction with a liquidity pool using the provided command-line arguments. We define it in multiple parts, as follows:
144. def
145. main
146. ():
147. """Run main."""
148. args
149. =
150. _parse_commandline
151. ()
152. **Define any wallet**
153. wallet
154. =
155. LocalWallet
156. .
157. generate
158. ()
159. **Network configuration**
160. ledger
161. =
162. LedgerClient
163. (NetworkConfig.
164. latest_stable_testnet
165. ())
166. **Add tokens to wallet**
167. faucet_api
168. =
169. FaucetApi

170. (NetworkConfig.
171. latest_stable_testnet
172. ())
173. faucet_api
174. .
175. get_wealth
176. (wallet.
177. address
178. ())
179. ## Define cw20, pair and liquidity token contracts
180. token_contract_address
181. =
182. (
183. "fetch1qr8ysysnfxmqzu7cu7cq7dsq5g2r0kvkg5e2wl2fnlkqss60hcjsxtljxl"
184.)
185. pair_contract_address
186. =
187. (
188. "fetch1vgnx2d46uvyxrg9pc5mktkcvkp4uflyp3j86v68pq4jxdc8j4y0s6ulf2a"
189.)
190. liq_token_contract_address
191. =
192. (
193. "fetch1alzhf9yhghud3qhucdjs895f3aek2egfq44qm0mfvahkv4jukx4qd0ltxx"
194.)
195. token_contract
196. =
197. LedgerContract
198. (
199. path
200. =
201. None
202. , client
203. =
204. ledger, address
205. =
206. token_contract_address
207.)
208. pair_contract
209. =
210. LedgerContract
211. (
212. path
213. =
214. None
215. , client
216. =
217. ledger, address
218. =
219. pair_contract_address
220.)
221. liq_token_contract
222. =
223. LedgerContract
224. (
225. path
226. =
227. None
228. , client
229. =
230. ledger, address
231. =
232. liq_token_contract_address
233.)
234. print

235. (
236. "Pool (initial state): "
237.)
238. print
239. (pair_contract.
240. query
241. ({
242. "pool"
243. : {}}),
244. "\n"
245.)
246. It starts by calling_parse_commandline()
247. to retrieve the command-line arguments. These arguments control various aspects of the liquidity pool interaction, like swap amounts and liquidity provision. We then create new wallet calledwallet
248. . This wallet will be used for conducting transactions. We proceed and set the network configuration to the latest stable testnet. Through thefaucet_api
249. we add tokens to the wallet. This simulates the process of acquiring tokens from an external source. We go on and define the contract addresses. In the part, addresses of three different contracts (CW20 token, pair, and liquidity token contracts) are defined. These contracts are essential for interacting with the liquidity pool. Finally we print the initial pool state. This provides an initial snapshot of the liquidity pool before any actions are taken.

250. Swap atestfet for CW20 tokens

251. swap_amount
252. =
253. str
254. (args.swap_amount)
255. native_denom
256. =
257. "atestfet"
258. tx
259. =
260. pair_contract
261. .
262. execute
263. (
264. {
265. "swap"
266. : {
267. "offer_asset"
268. : {
269. "info"
270. : {
271. "native_token"
272. : {
273. "denom"
274. : native_denom}},
275. "amount"
276. : swap_amount,
277. }
278. }
279. },
280. sender
281. =
282. wallet,
283. funds
284. =
285. swap_amount
286. +
287. native_denom,
288.)
289. print
290. (
291. f
292. "Swapping
293. {
294. swap_amount

295. +
296. native_denom
297. }
298. for CW20 Tokens..."
299.)
300. tx
301. .
302. wait_to_complete
303. ()
304. print
305. (
306. "Pool (after swap): "
307.)
308. print
309. (pair_contract.
310. query
311. ({
312. "pool"
313. : {}}),
314. "\n"
315.)
316. **To provide cw20 token to LP, increase your allowance first**
317. cw20_liquidity_amount
318. =
319. str
320. (args.cw20_liquidity_amount)
321. native_liquidity_amount
322. =
323. str
324. (args.native_liquidity_amount)
325. tx
326. =
327. token_contract
328. .
329. execute
330. (
331. {
332. "increase_allowance"
333. : {
334. "spender"
335. : pair_contract_address,
336. "amount"
337. : cw20_liquidity_amount,
338. "expires"
339. : {
340. "never"
341. : {}},
342. }
343. },
344. wallet,
345.)
346. print
347. (
348. "Increasing Allowance..."
349.)
350. tx
351. .
352. wait_to_complete
353. ()
354. In this part of the main() function, the script swaps a specified amount of atestfet tokens for CW20 tokens using thepair_contract
355. . This is done by constructing a transaction with the"swap"
356. operation.swap_amount

357. is the amount of atestfet tokens to swap, retrieved from the command-line arguments.native_denom
358. is set to"atestfet"
359. which is the native token denomination. The transaction is executed with theexecute()
360. method, specifying the"swap"
361. operation. Thesender
362. parameter is set to the user'swallet
363. , and thefunds
364. parameter is set to the amount being swapped in addition to the native denomination. The script then waits for the transaction to complete, and after this, a message is printed to indicate the swap operation has occurred. Within the function, we then provide CW20 tokens to the liquidity pool. The script first increases the allowance for the pair contract to spend CW20 tokens from the user's wallet. Thecw20_liquidity_amount
365. is the amount of CW20 tokens to provide to the LP, retrieved from the command-line arguments. Thenative_liquidity_amount
366. is the amount of atestfet tokens to provide to the LP, also retrieved from the command-line arguments. A transaction is created with the"increase_allowance"
367. operation using theexecute()
368. method. The transaction specifies thespender
369. (pair_contract_address
370.), theamount
371. to allow spending (cw20_liquidity_amount
372.), and anexpires
373. parameter set tonever
374. . The script waits for the transaction to complete, and after this, a message is printed to indicate that the allowance has been increased.

375. Provide Liquidity

376. Liquidity should be added so that the slippage tolerance parameter isn't exceeded

377. tx
378. =
379. pair_contract
380. .
381. execute
382. (
383. {
384. "provide_liquidity"
385. : {
386. "assets"
387. : [
388. {
389. "info"
390. : {
391. "token"
392. : {
393. "contract_addr"
394. : token_contract_address}},
395. "amount"
396. : cw20_liquidity_amount,
397. },
398. {
399. "info"
400. : {
401. "native_token"
402. : {
403. "denom"
404. : native_denom}},
405. "amount"
406. : native_liquidity_amount,
407. },
408.],
409. "slippage_tolerance"
410. :
411. "0.1"

412. ,
413. }
414. },
415. sender
416. =
417. wallet,
418. funds
419. =
420. native_liquidity_amount
421. +
422. native_denom,
423.)
424. print
425. (
426. f
427. "Providing
428. {
429. native_liquidity_amount
430. +
431. native_denom
432. }
433. and
434. {
435. cw20_liquidity_amount
436. }
437. CW20 tokens to Liquidity Pool..."
438.)
439. tx
440. .
441. wait_to_complete
442. ()
443. print
444. (
445. "Pool (after providing liquidity): "
446.)
447. print
448. (pair_contract.
449. query
450. ({
451. "pool"
452. : {}}),
453. "\n"
454.)

455. Withdraw Liquidity

456. LP_token_balance
457. =
458. liq_token_contract
459. .
460. query
461. (
462. {
463. "balance"
464. : {
465. "address"
466. :
467. str
468. (wallet.
469. address
470. ())}}
471.)
472. [
473. "balance"
474.]
475. withdraw_msg
476. =

```
'{"withdraw_liquidity":
{}
}'
withdraw_msg_bytes
=
withdraw_msg
.
encode
(
"ascii"
)
withdraw_msg_base64
=
base64
.
b64encode
(withdraw_msg_bytes)
msg
=
str
(withdraw_msg_base64)
[
2
:
-
1
]
tx
=
liq_token_contract
.
execute
(
{
"send"
: {
"contract"
: pair_contract_address,
"amount"
: LP_token_balance,
"msg"
: msg,
}
},
sender
=
wallet,
)
print
(
f
"Withdrawing
{
LP_token_balance
}
from pool's total share..."
)
tx
.
wait_to_complete
()
print
(
"Pool (after withdrawing liquidity): "
)
print
(pair_contract.
query
```

545. ({
546. "pool"
547. : {}}),
548. "\n"
549.)
550. if
551. **name**
552. ==
553. "**main**"
554. :
555. main
556. ()
557. Within themain()
558. script we would need to provide liquidity to the pool, ensuring that the slippage tolerance parameter isn't exceeded. Liquidity is added by creating a transaction with the"provide_liquidity"
559. operation. The assets being provided include CW20 tokens and atestfet tokens. These are specified in a list within the"assets"
560. field of the operation. The script also sets a slippage tolerance of0.1
561. , meaning that the price impact of the liquidity provision must be within 10% of the expected value. The transaction is executed withexecute()
562. method, specifying the"provide_liquidity"
563. operation. Thesender
564. parameter is set to the user's wallet, and thefunds
565. parameter includes the amount of atestfet tokens being provided. A message is printed indicating the amount of CW20 and atestfet tokens being provided to the liquidity pool.
566. Afterwards, the script initiates a withdrawal of liquidity from the pool. This involves creating a transaction with the"withdraw_liquidity"
567. operation. The LP token balance is queried usingquery()
568. method to determine the amount of LP tokens held by the user. A withdrawal message is constructed in JSON format and then encoded and base64-encoded to be included in the transaction. The transaction is executed with theexecute()
569. , specifying the"send"
570. operation. Thecontract
571. parameter is set to the pair contract address, theamount
572. parameter is set to the LP token balance, and themsg
573. parameter includes the withdrawal message. A message is printed indicating the amount of LP tokens being withdrawn from the pool. Also, the LP balance is printed after withdrawal take place.
574. In summary, the main function orchestrates a series of actions, simulating interactions with a liquidity pool. These actions include swapping tokens, providing liquidity, and withdrawing liquidity, and the state of the pool is printed at different stages to provide feedback to the user.
575. Save the script.

The overall script should be as follows:

aerial_liquidity_pool.py import argparse import base64 from cosmpy . aerial . client import LedgerClient , NetworkConfig from cosmpy . aerial . contract import LedgerContract from cosmpy . aerial . faucet import FaucetApi from cosmpy . aerial . wallet import LocalWallet

def

_parse_commandline (): parser = argparse . ArgumentParser () parser . add_argument ("swap_amount" , type = int , nargs = "?" , default = 10000 , help = "atestfet swap amount to get some cw20 tokens on wallet's address" ,) parser . add_argument ("cw20_liquidity_amount" , type = int , nargs = "?" , default = 100 , help = "amount of cw20 tokens that will be provided to LP" ,) parser . add_argument ("native_liquidity_amount" , type = int , nargs = "?" , default = 2470 , help = "amount of atestfet tokens that will be provided to LP" ,) return parser . parse_args ()

def

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

_parse_commandline ()

Define any wallet

wallet

LocalWallet . generate ()

Network configuration

ledger

LedgerClient (NetworkConfig. latest_stable_testnet ())

Add tokens to wallet

faucet_api

FaucetApi (NetworkConfig. latest_stable_testnet ()) faucet_api . get_wealth (wallet. address ())

Define cw20, pair and liquidity token contracts

token_contract_address

("fetch1qr8ysysnfxmqzu7cu7cq7dsq5g2r0kvkg5e2wl2fnlkqss60hcjsxtljxl") pair_contract_address = ("fetch1vgnx2d46uvyxrg9pc5mktkcvkp4uflyp3j86v68pq4jxdc8j4y0s6ulf2a") liq_token_contract_address = ("fetch1alzhf9yhghud3qhucdjs895f3aek2egfq44qm0mfvahkv4jukx4qd0ltxx")

token_contract

LedgerContract (path = None , client = ledger, address = token_contract_address) pair_contract =

LedgerContract (path = None , client = ledger, address = pair_contract_address) liq_token_contract =

LedgerContract (path = None , client = ledger, address = liq_token_contract_address)

print ("Pool (initial state): ") print (pair_contract. query ({ "pool" : {}}), "\n")

Swap atestfet for CW20 tokens

swap_amount

str (args.swap_amount) native_denom =

"atestfet"

tx

pair_contract . execute ({ "swap" : { "offer_asset" : { "info" : { "native_token" : { "denom" : native_denom}}, "amount" : swap_amount, } } }, sender = wallet, funds = swap_amount + native_denom,)

print (f "Swapping { swap_amount + native_denom } for CW20 Tokens...") tx . wait_to_complete ()

print ("Pool (after swap): ") print (pair_contract. query ({ "pool" : {}}), "\n")

To provide cw20 token to LP, increase your allowance first

cw20_liquidity_amount

str (args.cw20_liquidity_amount) native_liquidity_amount =

str (args.native_liquidity_amount)

tx

token_contract . execute ({ "increase_allowance" : { "spender" : pair_contract_address, "amount" : cw20_liquidity_amount, "expires" : { "never" : {}}, } }, wallet,)

print ("Increasing Allowance...") tx . wait_to_complete ()

Provide Liquidity

Liquidity should be added so that the slippage tolerance parameter isn't exceeded

tx

pair_contract . execute ({ "provide_liquidity" : { "assets" : [{ "info" : { "token" : { "contract_addr" : token_contract_address}}, "amount" : cw20_liquidity_amount, }, { "info" : { "native_token" : { "denom" : native_denom}}, "amount" : native_liquidity_amount, },], "slippage_tolerance" : "0.1" , } }, sender = wallet, funds = native_liquidity_amount + native_denom,)

print (f "Providing { native_liquidity_amount + native_denom } and { cw20_liquidity_amount } CW20 tokens to Liquidity Pool...") tx . wait_to_complete ()

print ("Pool (after providing liquidity): ") print (pair_contract. query ({ "pool" : {}}), "\n")

Withdraw Liquidity

LP_token_balance

liq_token_contract . query ({ "balance" : { "address" : str (wallet. address ())}}) ["balance"]

withdraw_msg

'{"withdraw_liquidity": {} }' withdraw_msg_bytes = withdraw_msg . encode ("ascii") withdraw_msg_base64 = base64 . b64encode (withdraw_msg_bytes) msg =

str (withdraw_msg_base64) [2 : - 1]

tx

liq_token_contract . execute ({ "send" : { "contract" : pair_contract_address, "amount" : LP_token_balance, "msg" : msg, } }, sender = wallet,)

print (f "Withdrawing { LP_token_balance } from pool's total share...") tx . wait_to_complete ()

print ("Pool (after withdrawing liquidity): ") print (pair_contract. query ({ "pool" : {}}), "\n")

if

name

==

"**main**" : main ()

Was this page helpful?

[Wallet top-up](#) [Swap automation](#)