Swap automation

Introduction

The following guide demonstrates an automated swapping strategy for a liquidity pool on the Fetch.ai network. It interacts with a liquidity pool contract and performs swaps between two different tokens (atestfet and CW20 tokens) based on specified price thresholds. A mean-reversion strategy expects the prices to return to "normal" levels or a certain moving average following a temporary price spike. We can construct a similar strategy using the Liquidity Pool, where we will set upper and lower bound prices that will trigger a sell and a buy transaction respectively. If the behavior of the LP prices works as expected always returning to a certain moving average, we could profit by selling high and buying low. We will do this by swapping atestfet and CW20 with the Liquidity Pool, we refer to a sell transaction when we sell atestfet and get CW20 tokens, a buy transaction would be exactly the opposite.

Walk-through

1. Let's start by creating a Python script and name it:touch aerial_swap_automation.py
2. Let's then import the needed classes:
3. import
4. argparse
5. from
6. time
7. import
8. sleep
9. from
10. cosmpy
11. .
12. aerial
13. .
14. client
15. import
16. LedgerClient
17. ,
18. NetworkConfig
19. from
20. cosmpy
21. .
22. aerial
23. .
24. contract
25. import
26. LedgerContract
27. from
28. cosmpy
29. .
30. aerial
31. .
32. faucet
33. import
34. FaucetApi
35. from
36. cosmpy
37. .
38. aerial
39. .
40. wallet
41. import
42. LocalWallet
43. We then need to define aswap_native_for_cw20()
44. function which performs a swap from native tokens (atestfet) to CW20 tokens within a liquidity pool:
45. def
46. swap_native_for_cw20
47. (
48. swap_amount
49. ,
50. pair_contract
51. ,

```
wallet
):
"""
Swap Native for cw20.
:param swap_amount: swap amount
:param pair_contract: pair contract address
:param wallet: wallet address
"""
tx
=
pair_contract
.
execute
(
{
"swap"
: {
"offer_asset"
: {
"info"
: {
"native_token"
: {
"denom"
:
"atestfet"
}},
"amount"
:
str
(swap_amount),
}
}
},
sender
=
wallet,
funds
=
str
(swap_amount)
+
"atestfet"
,
)
print
(
"swapping native for cw20 tokens"
)
tx
.
wait_to_complete
()
```

Within the function, we defined the following parameters:

- swap_amount
- : this parameter specifies the amount of native tokens to be swapped for CW20 tokens.
- pair_contract
- : this parameter represents the contract address of the liquidity pool pair where the swap will occur.
- wallet
- : this parameter represents the wallet address that will perform the swap.

The function constructs a transaction to execute the swap operation. Theexecute() method is called on thepair_contract

114. with a dictionary specifying the"swap"
115. operation. Inside the"swap"
116. operation, theoffer_asset
117. field is set to the following:
118.
    - info
119.
    - : this field specifies that the swap involves native tokens (native_token
120.
    -) with the denomination"atestfet"
121.
    - .
122.
    - amount
123.
    - : this field specifies the amount of native tokens to be swapped, which is converted to a string.
124. Thesender
125. parameter is set to thewallet
126. address, indicating that the wallet will initiate the swap. Thefunds
127. parameter is set to a string representing the total amount of funds being used for the swap, which includes theswap_amount
128. and"atestfet"
129. . Finally, the function waits for the transaction to complete and prints a message indicating that native tokens are being swapped for CW20 tokens.
130. We then need to define aswap_cw20_for_native()
131. function which performs a swap from CW20 tokens to native tokens (atestfet) within a liquidity pool:
132. def
133. swap_cw20_for_native
134. (
135. swap_amount
136. ,
137. pair_contract_address
138. ,
139. token_contract
140. ,
141. wallet
142.):
143. """
144. Swap cw20 for native.
145. :param swap_amount: swap amount
146. :param pair_contract_address: pair contract address
147. :param token_contract: token contract
148. :param wallet: wallet address
149. """
150. tx
151. =
152. token_contract
153. .
154. execute
155. (
156. {
157. "send"
158. : {
159. "contract"
160. : pair_contract_address,
161. "amount"
162. :
163. str
164. (swap_amount),
165. "msg"
166. :
167. "eyJzd2FwIjp7fX0="
168. ,
169. }
170. },
171. wallet,
172.)
173. print

174. (
175. "swapping cw20 for native tokens"
176.)
177. tx
178. .
179. wait_to_complete
180. ()
181. Within the function, we defined the following parameters:
182.
    - swap_amount
183.
    - : this parameter specifies the amount of CW20 tokens to be swapped for native tokens.
184.
    - pair_contract_address
185.
    - : this parameter represents the contract address of the liquidity pool pair where the swap will occur.
186.
    - token_contract
187.
    - : this parameter represents the contract for the CW20 token.
188.
    - wallet
189.
    - : This parameter represents the wallet address that will perform the swap.
190. The function constructs a transaction to execute the swap operation: theexecute()
191. method is called on thetoken_contract
192. with a dictionary specifying the"send"
193. operation. Inside this operation, the contract field is set topair_contract_address
194. , indicating that the CW20 tokens will be sent to the liquidity pool. Theamount
195. field is set to theswap_amount
196. , which is converted to a string. Themsg
197. field is set to the base64 encoded message"eyJzd2FwIjp7fX0="
198. , which likely contains additional instructions or parameters for the swap. Thewallet
199. address is specified as the sender of the transaction. Finally, the function waits for the transaction to complete and prints a message indicating that CW20 tokens are being swapped for native tokens.
200. We now would need to proceed by defining a_parse_commandline()
201. function:
202. def
203. _parse_commandline
204. ():
205. """Commandline parser."""
206. parser
207. =
208. argparse
209. .
210. ArgumentParser
211. ()
212. parser
213. .
214. add_argument
215. (
216. "trading_wallet"
217. ,
218. type
219. =
220. int
221. ,
222. nargs
223. =
224. "?"
225. ,
226. default
227. =
228. 1000000
229. ,
230. help
231. =
232. "initial atestfet balance to perform swaps using the liquidity pool"

```
,
)
parser
.
add_argument
(
"upper_bound"
,
type
=
int
,
nargs
=
"?"
,
default
=
20.5
,
help
=
"price upper bound that will trigger a swap from cw20 to native tokens"
,
)
parser
.
add_argument
(
"lower_bound"
,
type
=
int
,
nargs
=
"?"
,
default
=
19.5
,
help
=
"price lower bound that will trigger a swap from native to cw20 tokens"
,
)
parser
.
add_argument
(
"commission"
,
type
=
int
,
nargs
=
"?"
,
default
=
0.003
,
help
=
```

301. "LP commission, for terraswap the default is 0.3%"
302. ,
303.)
304. parser
305. .
306. add_argument
307. (
308. "interval_time"
309. ,
310. type
311. =
312. int
313. ,
314. nargs
315. =
316. "?"
317. ,
318. default
319. =
320. 5
321. ,
322. help
323. =
324. "interval time in seconds to query liquidity pool price"
325. ,
326.)
327. return
328. parser
329. .
330. parse_args
331. ()
332. This function is responsible for parsing command line arguments in the script. It uses theargparse.ArgumentParser()
333. class to define and handle the expected command line arguments:
334.
    - trading_wallet
335.
    - : this argument represents the initial balance of atestfet in the trading wallet. It's an optional argument, and if not provided, it defaults to1000000
336.
    - .
337.
    - upper_bound
338.
    - : this argument specifies the upper price threshold that will trigger a swap from cw20 to native tokens . If not provided, it defaults to20.5
339.
    - .
340.
    - lower_bound
341.
    - : this argument sets the lower price threshold that will trigger a swap from native to cw20 tokens. It defaults to19.5
342.
    - if not provided.
343.
    - commission
344.
    - : this argument defines the commission rate for the liquidity pool. The default is0.003
345.
    - , representing 0.3%.
346.
    - interval_time
347.
    - : this argument determines the interval (in seconds) at which the script queries the liquidity pool price. If not provided, it defaults to5
348.
    - seconds.
349. The function then returns an object containing the parsed arguments. These arguments can be accessed later in the

script to control the behavior of the swap automation.

350. We are ready to write down ourmain()
351. function:
352. def
353. main
354. ():
355. """Run main."""
356. args
357. =
358. _parse_commandline
359. ()
360. **Define any wallet**
361. wallet
362. =
363. LocalWallet
364. .
365. generate
366. ()
367. **Network configuration**
368. ledger
369. =
370. LedgerClient
371. (NetworkConfig.
372. latest_stable_testnet
373. ())
374. **Add tokens to wallet**
375. faucet_api
376. =
377. FaucetApi
378. (NetworkConfig.
379. latest_stable_testnet
380. ())
381. wallet_balance
382. =
383. ledger
384. .
385. query_bank_balance
386. (wallet.
387. address
388. ())
389. while
390. wallet_balance
391. <
392. (
393. 10
394. **
395. 18
396.)
397. :
398. print
399. (
400. "Providing wealth to wallet..."
401.)
402. faucet_api
403. .
404. get_wealth
405. (wallet.
406. address
407. ())

408. wallet_balance
409. =
410. ledger
411. .
412. query_bank_balance
413. (wallet.
414. address
415. ())

416. Define cw20, pair and liquidity token contracts

417. token_contract_address
418. =
419. (
420. "fetch1qr8ysysnfxmqzu7cu7cq7dsq5g2r0kvkg5e2wl2fnlkqss60hcjsxtljxl"
421.)
422. pair_contract_address
423. =
424. (
425. "fetch1vgnx2d46uvyxrg9pc5mktkcvkp4uflyp3j86v68pq4jxdc8j4y0s6ulf2a"
426.)
427. token_contract
428. =
429. LedgerContract
430. (
431. path
432. =
433. None
434. , client
435. =
436. ledger, address
437. =
438. token_contract_address
439.)
440. pair_contract
441. =
442. LedgerContract
443. (
444. path
445. =
446. None
447. , client
448. =
449. ledger, address
450. =
451. pair_contract_address
452.)

453. tokens in trading wallet (currency will vary [atestfet,cw20])

454. currency
455. =
456. "atestfet"
457. tokens
458. =
459. args
460. .
461. trading_wallet

462. Swap thresholds

463. upper_bound
464. =

465. args
466. .
467. upper_bound
468. lower_bound
469. =
470. args
471. .
472. lower_bound

473. LP commission

474. commission
475. =
476. args
477. .
478. commission

479. Wait time

480. interval
481. =
482. args
483. .
484. interval_time
485. while
486. True
487. :

488. Query LP status

489. pool
490. =
491. pair_contract
492. .
493. query
494. ({
495. "pool"
496. : {}})
497. native_amount
498. =
499. int
500. (pool[
501. "assets"
502.][
503. 1
504.][
505. "amount"
506.])
507. cw20_amount
508. =
509. int
510. (pool[
511. "assets"
512.][
513. 0
514.][
515. "amount"
516.])
517. if
518. currency
519. ==
520. "atestfet"
521. :

522. Calculate received cw20 tokens if atestfet tokens are

522. Calculate received cw20 tokens if atestfet tokens are given to LP

523. tokens_out
524. =
525. round
526. (
527. ((cw20_amount
528. *
529. tokens)
530. /
531. (native_amount
532. +
533. tokens))
534. *
535. (
536. 1
537. -
538. commission)
539.)

540. Sell price of atestfet => give atestfet, get cw20

541. sell_price
542. =
543. tokens
544. /
545. tokens_out
546. print
547. (
548. "atestfet sell price: "
549. , sell_price)
550. if
551. sell_price
552. <=
553. lower_bound
554. :
555. swap_native_for_cw20
556. (tokens, pair_contract, wallet)
557. tokens
558. =
559. int
560. (
561. token_contract.
562. query
563. (
564. {
565. "balance"
566. : {
567. "address"
568. :
569. str
570. (wallet.
571. address
572. ())}}
573.)[
574. "balance"
575.]
576.)

577. Trading wallet currency changed to cw20

578. currency
579. =
580. "CW20"

581. else
582. :

583. Calculate received atestfet tokens if cw20 tokens are given to LP

584. tokens_out
585. =
586. round
587. (
588. ((native_amount
589. *
590. tokens)
591. /
592. (cw20_amount
593. +
594. tokens))
595. *
596. (
597. 1
598. -
599. commission)
600.)

601. Buy price of atestfet => give cw20, get atestfet

602. buy_price
603. =
604. tokens_out
605. /
606. tokens
607. print
608. (
609. "atestfet buy price: "
610. , buy_price)
611. if
612. buy_price
613. =
614. upper_bound
615. :
616. swap_cw20_for_native
617. (
618. tokens, pair_contract_address, token_contract, wallet
619.)
620. tokens
621. =
622. tokens_out

623. Trading wallet currency changed to cw20

624. currency
625. =
626. "atestfet"
627. sleep
628. (interval)
629. if
630. **name**
631. ==
632. "**main**"
633. :
634. main
635. ()

636. Within themain()
637. function, the_parse_commandline()
638. function is used to parse command line arguments. It sets various parameters such as the initial trading wallet balance, upper and lower price bounds for triggering swaps, liquidity pool commission, and interval time for querying the liquidity pool price, and all of these values are store in theargs
639. variable. After this, a new wallet is generated using thegenerate()
640. method of theLocalWallet
641. class, and network configuration is set up using theLedgerClient()
642. class. Tokens are added to the wallet by using the Faucet API. This happens within awhile
643. loop which continues until the wallet balance reaches at least10**18
644. . The wallet balance is retrieved using thequery_bank_balance()
645. . Afterwards, we need to define the addresses of the CW20, pair, and liquidity token contracts, as well as initialise various variables based on the command line arguments, including the initial wallet balance,upper_bound
646. andlower_bound
647. price bounds for swaps, LP commission rate, and the interval at which to check the liquidity pool price.
648. We then define a loop (while True
649.), which:
650.
    - Queries the liquidity pool status (pair_contract.query({"pool": {}})
651.
    -) to get the current amounts of native tokens (atestfet
652.
    -) and CW20 tokens.
653.
    - Checks the current currency in the trading wallet (currency
654.
    -), which can be either native or CW20 tokens.
655.
    - If the currentcurrency
656.
    - isatestfet
657.
    - , it calculates the potential amount of CW20 tokens that would be received if native tokens were given to the liquidity pool. This is done based on the ratio of CW20 tokens to the total of native tokens and current wallet tokens, with a deduction for the LP commission. It calculates asell_price
658.
    - as the ratio of the current wallet tokens to tokens swapped out.
659.
    - If the sell price is lower than or equal to the specifiedlower_bound
660.
    - , it triggers theswap_native_for_cw20()
661.
    - function, which swaps atestfet tokens for CW20 tokens.
662.
    - After the successful swap, it updates the tokens variable to the new balance of CW20 tokens and changes the currency to"CW20"
663.
    - .
664.
    - If the current currency is"CW20"
665.
    - , it calculates the potential amount of atestfet tokens that would be received if CW20 tokens are given to the liquidity pool. This is done based on the ratio of native tokens to the total of CW20 tokens and current wallet tokens, with a deduction for the LP commission. It calculates abuy_price
666.
    - as the ratio of potential atestfet tokens to the current wallet tokens.
667.
    - If thebuy_price
668.
    - is higher than or equal to the specifiedupper_bound
669.
    - , it triggers theswap_cw20_for_native()
670.
    - function, which swaps CW20 tokens for atestfet tokens.
671.
    - After the successful swap, it updates the tokens variable to the new balance of atestfet tokens and changes the currency to"atestfet"
672.

- .

673.

- The loop then waits for the specifiedinterval

674.

- before checking the liquidity pool status and performing the next iteration.

675. Save the script.

The overall script should be as follows:

aerial_swap_automation.py import argparse from time import sleep

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

swap_native_for_cw20 (swap_amount ,

pair_contract ,

wallet): """ Swap Native for cw20.

:param swap_amount: swap amount :param pair_contract: pair contract address :param wallet: wallet address

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

"atestfet" ,) print ("swapping native for cw20 tokens") tx . wait_to_complete ()

def

swap_cw20_for_native (swap_amount ,

pair_contract_address ,

token_contract ,

wallet): """ Swap cw20 for native.

:param swap_amount: swap amount :param pair_contract_address: pair contract address :param token_contract: token contract :param wallet: wallet address

""" tx = token_contract . execute ({ "send" : { "contract" : pair_contract_address, "amount" : str (swap_amount), "msg" : "eyJzd2FwIjp7fX0=" , } }, wallet,) print ("swapping cw20 for native tokens") tx . wait_to_complete ()

def

_parse_commandline (): """Commandline parser.""" parser = argparse . ArgumentParser ()

parser . add_argument ("trading_wallet" , type = int , nargs = "?" , default = 1000000 , help = "initial atestfet balance to perform swaps using the liquidity pool" ,) parser . add_argument ("upper_bound" , type = int , nargs = "?" , default = 20.5 , help = "price upper bound that will trigger a swap from cw20 to native tokens" ,) parser . add_argument ("lower_bound" , type = int , nargs = "?" , default = 19.5 , help = "price lower bound that will trigger a swap from native to cw20 tokens" ,) parser . add_argument ("commission" , type = int , nargs = "?" , default = 0.003 , help = "LP commission, for terraswap the default is 0.3%" ,) parser . add_argument ("interval_time" , type = int , nargs = "?" , default = 5 , help = "interval time in seconds to query liquidity pool price" ,)

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 ())

wallet_balance

ledger . query_bank_balance (wallet. address ())

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

Define cw20, pair and liquidity token contracts

token_contract_address

("fetch1qr8ysysnfxmqzu7cu7cq7dsq5g2r0kvkg5e2wl2fnlkqss60hcjsxtljxl") pair_contract_address = ("fetch1vgnx2d46uvyxrg9pc5mktkcvkp4uflyp3j86v68pq4jxdc8j4y0s6ulf2a")

token_contract

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

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

tokens in trading wallet (currency will vary [atestfet,cw20])

currency

"atestfet" tokens = args . trading_wallet

Swap thresholds

upper_bound

args . upper_bound lower_bound = args . lower_bound

LP commission

commission

args . commission

Wait time

interval

args . interval_time

while

True :

Query LP status

pool

pair_contract . query ({ "pool" : {}}) native_amount =

int (pool["assets"][1]["amount"]) cw20_amount =

int (pool["assets"][0]["amount"])

if currency ==

"atestfet" :

Calculate received cw20 tokens if atestfet tokens are given to LP

tokens_out

round (((cw20_amount * tokens) / (native_amount + tokens)) * (1

- commission))

Sell price of atestfet => give atestfet, get cw20

sell_price

tokens / tokens_out print ("atestfet sell price: " , sell_price) if sell_price <= lower_bound : swap_native_for_cw20 (tokens, pair_contract, wallet) tokens =

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

Trading wallet currency changed to cw20

currency

"CW20" else :

Calculate received atestfet tokens if cw20 tokens are given to LP

tokens_out

round (((native_amount * tokens) / (cw20_amount + tokens)) * (1

- commission))

Buy price of atestfet => give cw20, get atestfet

buy_price

tokens_out / tokens print ("atestfet buy price: " , buy_price) if buy_price

> = upper_bound : swap_cw20_for_native (tokens, pair_contract_address, token_contract, wallet) tokens = tokens_out

Trading wallet currency changed to cw20

currency

"atestfet"

sleep (interval)

if

name

==

"**main**" : main ()

Was this page helpful?

Liquidity pool Installation