

Oracles

Introduction

Oracles are entities that can update state variables in smart contracts and whose goal is usually to accurately estimate or predict some real world quantity or quantities. These quantities can then be used in the logic of other smart contracts. This guide shows how to write a CosmPy script that deploys and updates an oracle contract with a coin price, and another script that deploys a contract that queries this coin price.

Walk-through

Here we provide an overview guide for setting up your own aerial oracle in few steps.

We initially need to download the binaries for both contracts, which can be done as follows:

```
wget https://raw.githubusercontent.com/fetchai/agents-aea/develop/packages/fetchai/contracts/oracle/build/oracle.wasm
wget https://raw.githubusercontent.com/fetchai/agents-aea/develop/packages/fetchai/contracts/oracle_client/build/oracle_client.wasm
```

Aerial oracle

1. First of all, create a Python script and name it:touch aerial_oracle.py
2. We would then also require the following imports:
3. from
4. time
5. import
6. sleep
7. import
8. requests
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. from
44. cosmpy
45. .

```

46. crypto
47. .
48. address
49. import
50. Address
51. We then need to choose a data source for the coin price, the update interval, the decimal precision, and the decimal
    timeout for the oracle value:
52. COIN_PRICE_URL
53. =
54. (
55. "https://api.coingecko.com/api/v3/simple/price?ids=fetch-ai&vs_currencies=usd"
56. )
57. UPDATE_INTERVAL_SECONDS
58. =
59. 10
60. ORACLE_VALUE_DECIMALS
61. =
62. 5
63. DEFAULT_TIMEOUT
64. =
65. 60.0
66. We then proceed and define a _parse_commandline()
67. function by first importing the argparse
68. module, which is a standard Python module for parsing command-line arguments:
69. def
70. _parse_commandline
71. ():
72. parser
73. =
74. argparse
75. .
76. ArgumentParser
77. ()
78. parser
79. .
80. add_argument
81. (
82. "contract_path"
83. , help
84. =
85. "The path to the oracle contract to upload"
86. )
87. parser
88. .
89. add_argument
90. (
91. "contract_address"
92. ,
93. nargs
94. =
95. "?"
96. ,
97. type
98. =
99. Address,
100. help
101. =
102. "The address of the oracle contract if already deployed"
103. ,
104. )
105. return
106. parser
107. .
108. parse_args
109. ()
110. This first creates an argumentparser
111. object. TheArgumentParser
112. class provides a way to specify the arguments your script should accept and automatically generates help messages

```

```

    and error messages. We then use add_argument()
113. to add a positional argument named contract_path
114. . This argument is required and should be a path to the oracle contract that you want to upload. The help argument
    provides a description of what this argument does. We further add another positional argument
    named contract_address
115. . This argument is optional (nargs="?"
116. allows it to be omitted), and it should be of type Address
117. . The type
118. argument specifies the type of the argument. In this case, Address
119. is a custom type or class used to represent addresses. The help
120. argument provides a description of what this argument does. At the end, we parse the command-line arguments
    provided when the script is executed. It returns an object that contains the values of the parsed arguments.
121. We then need to proceed and define our main()
122. function:
123. def
124. main
125. ():
126. """Run main."""
127. args
128. =
129. _parse_commandline
130. ()
131. wallet
132. =
133. LocalWallet
134. .
135. generate
136. ()
137. ledger
138. =
139. LedgerClient
140. (NetworkConfig.
141. fetchai_stable_testnet
142. ())
143. faucet_api
144. =
145. FaucetApi
146. (NetworkConfig.
147. fetchai_stable_testnet
148. ())
149. wallet_balance
150. =
151. ledger
152. .
153. query_bank_balance
154. (wallet.
155. address
156. ())
157. while
158. wallet_balance
159. <
160. (
161. 10
162. **
163. 18
164. )
165. :
166. print
167. (
168. "Providing wealth to wallet..."
169. )
170. faucet_api
171. .
172. get_wealth
173. (wallet.
174. address
175. ())
176. wallet_balance

```

```
177. =
178. ledger
179. .
180. query_bank_balance
181. (wallet.
182. address
183. ())
184. contract
185. =
186. LedgerContract
187. (args.contract_path, ledger, address
188. =
189. args.contract_address)
190. if
191. not
192. args
193. .
194. contract_address
195. :
196. instantiation_message
197. =
198. {
199. "fee"
200. :
201. "100"
202. }
203. contract
204. .
205. deploy
206. (instantiation_message, wallet, funds
207. =
208. "1atestfet"
209. )
210. print
211. (
212. f
213. "Oracle contract deployed at:
214. {
215. contract.address
216. }
217. "
218. )
219. grant_role_message
220. =
221. {
222. "grant_oracle_role"
223. :
224. {
225. "address"
226. :
227. wallet
228. }}
229. contract
230. .
231. execute
232. (grant_role_message, wallet).
233. wait_to_complete
234. ()
235. print
236. (
237. f
238. "Oracle role granted to address:
239. {
240. wallet
241. }
242. "
243. )
244. while
```

```
245. True
246. :
247. resp
248. =
249. requests
250. .
251. get
252. (COIN_PRICE_URL, timeout
253. =
254. DEFAULT_TIMEOUT).
255. json
256. ()
257. price
258. =
259. resp
260. [
261. "fetch-ai"
262. ]
263. [
264. "usd"
265. ]
266. value
267. =
268. int
269. (price
270. *
271. 10
272. **
273. ORACLE_VALUE_DECIMALS)
274. update_message
275. =
276. {
277. "update_oracle_value"
278. :
279. {
280. "value"
281. :
282. str
283. (value),
284. "decimals"
285. :
286. str
287. (ORACLE_VALUE_DECIMALS),
288. }
289. }
290. contract
291. .
292. execute
293. (update_message, wallet).
294. wait_to_complete
295. ()
296. print
297. (
298. f
299. "Oracle value updated to:
300. {
301. price
302. }
303. USD"
304. )
305. print
306. (
307. f
308. "Next update in
309. {
310. UPDATE_INTERVAL_SECONDS
311. }
312. seconds..."
```

```

313. )
314. sleep
315. (UPDATE_INTERVAL_SECONDS)
316. if
317. name
318. ==
319. "main"
320. :
321. main
322. ()
323. This defines ourmain()
324. function. When we run the script, the code inside main()
325. will be executed. args = _parse_commandline()
326. calls the _parse_commandline()
327. function that we defined earlier. It parses the command-line arguments and returns an object (args
328. ) containing the values of the parsed arguments. We then generate a new local wallet, and then create a client for
    interacting with a blockchain ledger, using LedgerClient()
329. class. We configured it to use the Fetch.ai stable testnet. We then create a client for interacting with a faucet API and
    query the balance of the wallet's address using the query_bank_balance()
330. method. We also define an initial while
331. loop which continues as long as the wallet_balance
332. is less than 10**18
333. . Inside this first loop: it prints a message indicating that wealth is being provided to the wallet, then it calls the faucet
    API to get wealth for the wallet, and it updates the wallet_balance
334. by querying the bank balance again.
335. After this, we create a contract
336. object using LedgerContract()
337. : this takes the path to the oracle contract file, the ledger client, and optionally, the contract address. if not
    args.contract_address:
338. condition checks if args.contract_address
339. is not provided. If it has not been provided, it means the contract has not been deployed yet. We then set up an
    instantiation message with a fee of 100. We can then deploy the contract using the provided instantiation message,
    the wallet, and a specified fund source ("latestfet")
340. in this case).
341. The print()
342. function prints the address of the deployed oracle contract. After this, we define a grant_role_message
343. object which sets up a message to grant the oracle role to the address associated with the wallet, and execute the
    message to grant the oracle role and wait for the transaction to complete. The following print()
344. function prints a message indicating that the oracle role has been granted to the address associated with the wallet.
345. We can finally define a second while
346. loop which runs indefinitely: it sends a GET request to a URL (COIN_PRICE_URL
347. ) to retrieve coin prices, then extracts the price in USD. It then calculates a value based on the price and the specified
    decimal precision (ORACLE_VALUE_DECIMALS
348. ), and sets up an update message with the new oracle value. Lastly, it executes the update message, waits for the
    transaction to complete, prints the updated oracle value and indicates when the next update will occur.

```

This script let us interact with a blockchain ledger, deploy a contract, and perform oracle-related tasks such as updating values based on external data.

1. Save the script.

The overall script should be as follows:

```
aerial_oracle.py import argparse from time import sleep
```

```
import requests
```

```
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 from cosmpy . crypto . address
import Address
```

COIN_PRICE_URL

```
( "https://api.coingecko.com/api/v3/simple/price?ids=fetch-ai&vs_currencies=usd" ) UPDATE_INTERVAL_SECONDS =
```

```
10 ORACLE_VALUE_DECIMALS =
```

```
5 DEFAULT_TIMEOUT =
```

60.0

def

```
_parse_commandline(): parser = argparse.ArgumentParser() parser.add_argument("contract_path", help="The path to the oracle contract to upload") parser.add_argument("contract_address", nargs="?", type=Address, help="The address of the oracle contract if already deployed",) return parser.parse_args()
```

def

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

```
_parse_commandline()
```

wallet

```
LocalWallet.generate()
```

ledger

```
LedgerClient(NetworkConfig.fetchai_stable_testnet()) faucet_api =
```

```
FaucetApi(NetworkConfig.fetchai_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())
```

contract

```
LedgerContract(args.contract_path, ledger, address=args.contract_address)
```

if

```
not args.contract_address: instantiation_message =
```

```
{ "fee":
```

```
"100" } contract.deploy(instantiation_message, wallet, funds="1atestfet")
```

```
print(f"Oracle contract deployed at: {contract.address}")
```

grant_role_message

```
{ "grant_oracle_role":
```

```
{ "address": wallet }} contract.execute(grant_role_message, wallet).wait_to_complete()
```

```
print(f"Oracle role granted to address: {wallet}")
```

while

```
True: resp = requests.get(COIN_PRICE_URL, timeout=DEFAULT_TIMEOUT).json() price = resp["fetch-ai"]["usd"]  
value =
```

```
int(price *
```

```
10 ** ORACLE_VALUE_DECIMALS)
```

update_message

```

{ "update_oracle_value" :
{ "value" :
str (value), "decimals" :
str (ORACLE_VALUE_DECIMALS), } } contract . execute (update_message, wallet). wait_to_complete ()

print ( f "Oracle value updated to: { price } USD" ) print ( f "Next update in { UPDATE_INTERVAL_SECONDS } seconds..." )
sleep (UPDATE_INTERVAL_SECONDS)

if

name

==

"main" : main ()

```

Oracle client

Now, we will write a script that deploys a contract that can request the oracle value in exchange for the required fee.

1. Let's first create a Python script and name it: `aerial_oracle_client.py`
2. We start by importing the needed classes and define `REQUEST_INTERVAL_SECONDS`
3. variable:
4. import
5. argparse
6. from
7. time
8. import
9. sleep
10. from
11. cosmpy
12. .
13. aerial
14. .
15. client
16. import
17. LedgerClient
18. ,
19. NetworkConfig
20. from
21. cosmpy
22. .
23. aerial
24. .
25. contract
26. import
27. LedgerContract
28. from
29. cosmpy
30. .
31. aerial
32. .
33. faucet
34. import
35. FaucetApi
36. from
37. cosmpy
38. .
39. aerial
40. .
41. wallet
42. import
43. LocalWallet
44. from
45. cosmpy
46. .
47. crypto


```

48. .
49. address
50. import
51. Address
52. REQUEST_INTERVAL_SECONDS
53. =
54. 10
55. Like before, we proceed and define a_parse_commandline()
56. function:
57. def
58. _parse_commandline
59. ():
60. parser
61. =
62. argparse
63. .
64. ArgumentParser
65. ()
66. parser
67. .
68. add_argument
69. (
70. "contract_path"
71. , help
72. =
73. "The path to the oracle client contract to upload"
74. )
75. parser
76. .
77. add_argument
78. (
79. "oracle_contract_address"
80. ,
81. type
82. =
83. Address,
84. help
85. =
86. "The address of the oracle contract"
87. ,
88. )
89. parser
90. .
91. add_argument
92. (
93. "contract_address"
94. ,
95. nargs
96. =
97. "?"
98. ,
99. type
100. =
101. Address,
102. help
103. =
104. "The address of the oracle client contract if already deployed"
105. ,
106. )
107. return
108. parser
109. .
110. parse_args
111. ()
112. This_parse_commandline()
113. function is designed to parse command-line arguments. We first create a parser
114. object. This object is used to specify what command-line arguments the program should expect. We then use
    the add_argument()

```

```

115. method to define the arguments that the program expects. In this function, there are three arguments being defined:
116.
    ◦ contract_path
117.
    ◦ : this is a required argument. It expects a string representing the path to the oracle client contract to upload.
118.
    ◦ oracle_contract_address
119.
    ◦ : this is also a required argument. It expects anAddress
120.
    ◦ object representing the address of the oracle contract.
121.
    ◦ contract_address
122.
    ◦ : this is an optional argument. It expects anAddress
123.
    ◦ object and is used to specify the address of the oracle client contract if it has already been deployed.
        Thenargs="?"
124.
    ◦ indicates that this argument is optional.
125. The function returns an object containing the parsed values.
126. We can now define ourmain()
127. function.
128. def
129. main
130. ():
131. """Run main."""
132. args
133. =
134. _parse_commandline
135. ()
136. wallet
137. =
138. LocalWallet
139. .
140. generate
141. ()
142. ledger
143. =
144. LedgerClient
145. (NetworkConfig.
146. fetchai_stable_testnet
147. ())
148. faucet_api
149. =
150. FaucetApi
151. (NetworkConfig.
152. fetchai_stable_testnet
153. ())
154. wallet_balance
155. =
156. ledger
157. .
158. query_bank_balance
159. (wallet.
160. address
161. ())
162. while
163. wallet_balance
164. <
165. (
166. 10
167. **
168. 18
169. )
170. :
171. print
172. (

```

```
173. "Providing wealth to wallet..."
174. )
175. faucet_api
176. .
177. get_wealth
178. (wallet.
179. address
180. ())
181. wallet_balance
182. =
183. ledger
184. .
185. query_bank_balance
186. (wallet.
187. address
188. ())
189. contract
190. =
191. LedgerContract
192. (args.contract_path, ledger, address
193. =
194. args.contract_address)
195. if
196. not
197. args
198. .
199. contract_address
200. :
201. instantiation_message
202. =
203. {
204. "oracle_contract_address"
205. :
206. str
207. (args.oracle_contract_address)
208. }
209. contract
210. .
211. deploy
212. (instantiation_message, wallet)
213. print
214. (
215. f
216. "Oracle client contract deployed at:
217. {
218. contract.address
219. }
220. "
221. )
222. while
223. True
224. :
225. request_message
226. =
227. {
228. "query_oracle_value"
229. :
230. {}}
231. contract
232. .
233. execute
234. (
235. request_message, wallet, funds
236. =
237. "100atestfet"
238. ).
239. wait_to_complete
240. ()
```

```

241. result
242. =
243. contract
244. .
245. query
246. ({
247. "oracle_value"
248. : {}})
249. print
250. (
251. f
252. "Oracle value successfully retrieved:
253. {
254. result
255. }
256. "
257. )
258. sleep
259. (REQUEST_INTERVAL_SECONDS)
260. if
261. name
262. ==
263. "main"
264. :
265. main
266. ()
267. The first line calls the _parse_commandline()
268. function that we defined earlier. It will parse the command-line arguments and return an object (args
269. ) containing the parsed values. We proceed and generate a new local wallet, wallet
270. , and then create a new ledger
271. object for interacting with the blockchain or ledger system, using LedgerClient()
272. . Afterwards, we create a FaucetApi
273. object, faucet_api
274. , which is used for interacting with the faucet service. We use the query_bank_balance()
275. method to query the balance associated with the wallet's address. We then define a while
276. loop which will continue as long as the wallet_balance
277. is less than 10**18
278. . This is to ensure the wallet has a sufficient balance. Afterwards, we use the get_wealth()
279. method to add wealth to the wallet, and then create a new LedgerContract()
280. object which takes the contract_path
281. , the ledger
282. object, and an optional contract_address
283. .if not args.contract_address:
284. checks if args.contract_address
285. is not provided. If it has not been provided, it means that the contract has not been deployed yet. We then create
  an instantiation_message
286. , which contains the data needed for deploying the contract. contract.deploy()
287. deploys the contract with the provided instantiation_message
288. and the wallet
289. . The code then prints out the address of the deployed contract. Finally, we define a second loop starting with while
  True:
290. which repeatedly executes the following steps:
291.
    ◦ It creates a request message, which is used to query the oracle value.
292.
    ◦ It executes the contract function call with the request message, using the wallet for authorization. The funds
      argument is set to "100atestfet"
293.
    ◦ .
294.
    ◦ It queries the contract for the oracle value.
295.
    ◦ It prints out the retrieved oracle value.
296.
    ◦ It finally waits for a specified number of seconds (defined by REQUEST_INTERVAL_SECONDS
297.
      ◦ ) before the next iteration. This is likely to prevent overloading the system with requests.
298. Save the script.

```

The overall script should be as follows:

```
aerial_oracle_client.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 from cosmpy . crypto . address  
import Address
```

REQUEST_INTERVAL_SECONDS

```
10
```

```
def
```

```
_parse_commandline (): parser = argparse . ArgumentParser () parser . add_argument ( "contract_path" , help = "The path  
to the oracle client contract to upload" ) parser . add_argument ( "oracle_contract_address" , type = Address, help = "The  
address of the oracle contract" , ) parser . add_argument ( "contract_address" , nargs = "?" , type = Address, help = "The  
address of the oracle client contract if already deployed" , ) return parser . parse_args ()
```

```
def
```

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

```
_parse_commandline ()
```

wallet

```
LocalWallet . generate ()
```

ledger

```
LedgerClient (NetworkConfig. fetchai_stable_testnet ()) faucet_api =
```

```
FaucetApi (NetworkConfig. fetchai_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 ())
```

contract

```
LedgerContract (args.contract_path, ledger, address = args.contract_address)
```

```
if
```

```
not args . contract_address : instantiation_message =
```

```
{ "oracle_contract_address" :
```

```
str (args.oracle_contract_address) } contract . deploy (instantiation_message, wallet)
```

```
print ( f "Oracle client contract deployed at: { contract.address } " )
```

```
while
```

```
True : request_message =
```

```
{ "query_oracle_value" :
```

```
}} contract . execute ( request_message, wallet, funds = "100atestfet" ). wait_to_complete ()
```

result

```
contract . query ({ "oracle_value" : {}}) print ( f "Oracle value successfully retrieved: { result } " )
```

```
sleep (REQUEST_INTERVAL_SECONDS)
```

```
if
```

```
name
```

```
==
```

"main" : main () Bear in mind that specific data related to the oracle's address and contract need to be provided by hand based on your personalized information!

Was this page helpful?

[Stake optimizer](#) [Wallet top-up](#)