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. .

crypto
.
address
import
Address
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:
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
We then proceed and define a_parse_commandline()
function by first importing theargparse
module, which is a standard Python module for parsing command-line arguments:
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
()
This first creates an argumentparser
object. TheArgumentParser
class provides a way to specify the arguments your script should accept and automatically generates help messages

and error messages. We then useadd_argument()
113. to add a positional argument namedcontract_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 namedcontract_address
115. . This argument is optional (nargs="?"
116. allows it to be omitted), and it should be of typeAddress
117. . Thetype
118. argument specifies the type of the argument. In this case,Address
119. is a custom type or class used to represent addresses. Thehelp
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 ourmain()
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

```
=
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..."
```

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 insidemain()
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, usingLedgerClient()
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 thequery_bank_balance()
330. method. We also define an initialwhile
331. loop which continues as long as thewallet_balance
332. is less than10**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 thewallet_balance
334. by querying the bank balance again.
335. After this, we create acontract
336. object usingLedgerContract()
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 ifargs.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 ("1atestfet"
340. in this case).
341. Theprint()
342. function prints the address of the deployed oracle contract. After this, we define agrant_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 followingprint()
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 secondwhile
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:touch aerial_oracle_client.py
2. We start by importing the needed classes and define aREQUEST_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

. address import Address REQUEST_INTERVAL_SECONDS = 10 Like before, we proceed and define a_parse_commandline() function: 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 () This_parse_commandline() function is designed to parse command-line arguments. We first create aparser object. This object is used to specify what command-line arguments the program should expect. We then use theadd_argument()

method to define the arguments that the program expects. In this function, there are three arguments being defined:

- contract_path
- : this is a required argument. It expects a string representing the path to the oracle client contract to upload.
- oracle_contract_address
- : this is also a required argument. It expects anAddress
- object representing the address of the oracle contract.
- contract_address
- : this is an optional argument. It expects anAddress
- object and is used to specify the address of the oracle client contract if it has already been deployed. Thenargs="?"
- indicates that this argument is optional.

The function returns an object containing the parsed values.
We can now define ourmain()
function.

```
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
()
The first line calls the_parse_commandline()
function that we defined earlier. It will parse the command-line arguments and return an object (args
) containing the parsed values. We proceed and generate a new local wallet,wallet
, and then create a newledger
object for interacting with the blockchain or ledger system, usingLedgerClient()
. Afterwards, we create aFaucetApi
object,faucet_api
, which is used for interacting with the faucet service. We use thequery_bank_balance()
method to query the balance associated with the wallet's address. We then define awhile
loop which will continue as long as thewallet_balance
is less than10**18
. This is to ensure the wallet has a sufficient balance. Afterwards, we use theget_wealth()
method to add wealth to the wallet, and then create a newLedgerContract()
object which takes thecontract_path
, theledger
object, and an optionalcontract_address
.if not args.contract_address:
checks ifargs.contract_address
is not provided. If it has not been provided, it means that the contract has not been deployed yet. We then create aninstantiation_message
, which contains the data needed for deploying the contract.contract.deploy()
deploys the contract with the providedinstantiation_message
and thewallet
. The code then prints out the address of the deployed contract. Finally, we define a second loop starting withwhile True:
which repeatedly executes the following steps:

- It creates a request message, which is used to query the oracle value.
- It executes the contract function call with the request message, using the wallet for authorization. The funds argument is set to"100atestfet"
- .
- It queries the contract for the oracle value.
- It prints out the retrieved oracle value.
- It finally waits for a specified number of seconds (defined byREQUEST_INTERVAL_SECONDS
-) before the next iteration. This is likely to prevent overloading the system with requests.

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