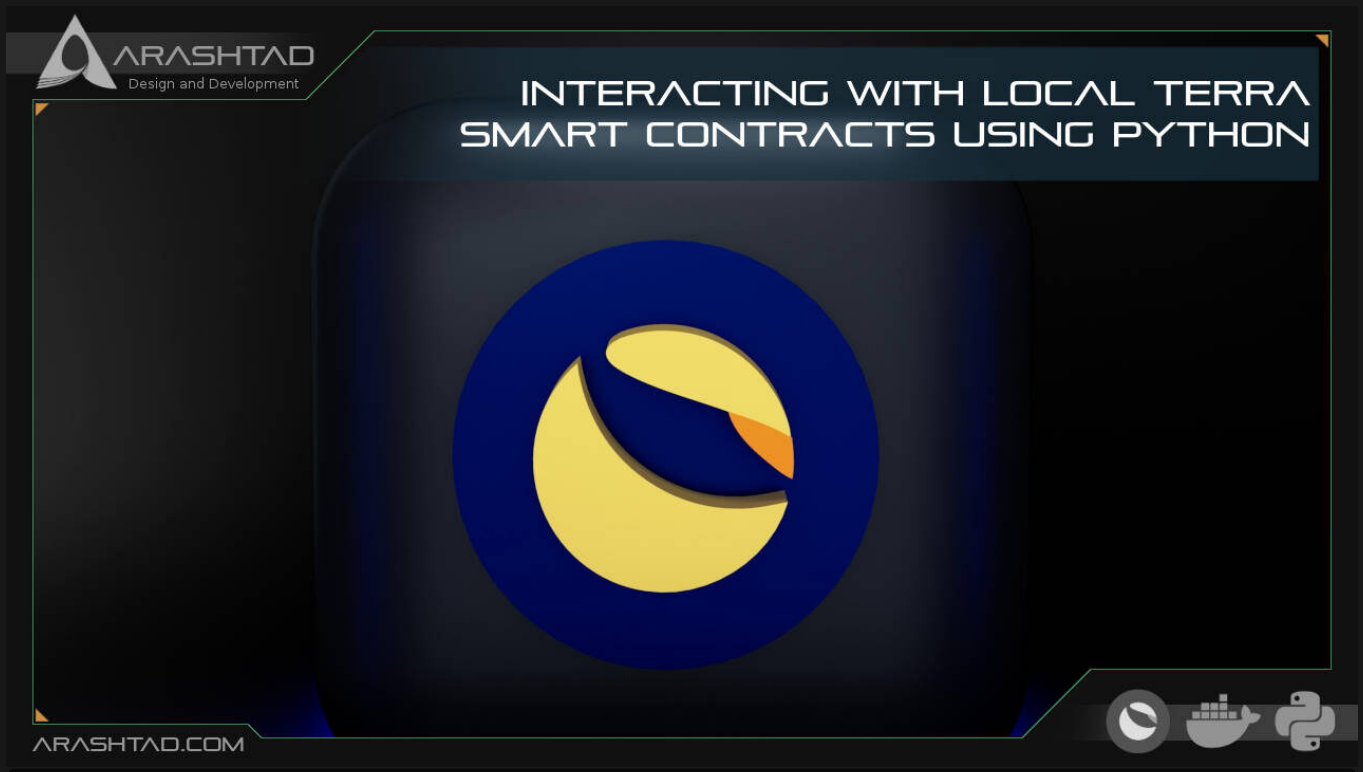
Interacting with Local Terra Smart Contracts Using Python

No comments



In this article, we first connect to local Terra using Docker. Then we try to interact with local Terra smart contracts using Python scripts and at the same time use the CosmWasm functions inside `LocalTerra\src` folder written in Rust programming language. Finally, we will manage to interact with it.

Connecting to Local Terra with Docker with Python

First off, To start our interaction with local Terra, we need to make sure docker is running: `sudo usermod -aG docker $USER` Make sure you sign in to your docker profile to be able to easily log in using the following command `docker login`. `docker-compose up` Result: `terrad_1 | 8:48PM INF received proposal module=consensus proposal={"Type":32,"block_id":{"hash":"3F119B6520D17CCC29A23F016B9E40151E3A5 parts": {"hash":"5928F59D217852E3722E46A9134D40F9329D540555EAD1DAE4B5EFA8A41E88D4","total":1}},"height":1948,"pol_round":-1,"round":0,"signature":"cYh80NzZezistnyNzEP2yFDnsoc7JOB+/8J1wnell-Gq76IE3qvH9jU2KvQx6u4y+A9xYrIQEJonzxa//vOZfBw==","timestamp":"2022-`

```
05-08T20:48:12.151658182Z"} terrad_1 | 8:48PM INF received complete
proposal block
hash=3F119B6520D17CCC29A23F016B9E40151E3A5784A3F83D65BD0DD2D282FB1A4F
height=1948 module=consensus terrad_1 | 8:48PM INF finalizing commit
of block
hash=3F119B6520D17CCC29A23F016B9E40151E3A5784A3F83D65BD0DD2D282FB1A4F
height=1948 module=consensus num_txs=0 root=332F90B4F0706BCB055547-
FA8443A1F623FC3DAE4043162730F59D15612D5FAE terrad_1 | 8:48PM INF
minted coins from module account amount=226578914uluna from=mint
module=x/bank terrad_1 | 8:48PM INF executed block height=1948
module=state num_invalid_txs=0 num_valid_txs=0 terrad_1 | 8:48PM INF
commit synced
commit=436F6D6D697449447B5B393820323020343020323436203134362031303
320363420323133203935203131322031313720313539203933203135362032313 12
036322033322038362031372031323520323339203231312035332033372031393 52
036372037392036352038302032353220322031383 55D3A3739437D terrad_1 |
8:48PM INF committed state
app_hash=621428F6926740D55F70759F5D9CD33E2056117DEFD33525C3434F41
50FC02B9 height=1948 module=state num_txs=0 terrad_1 | 8:48PM INF
indexed block height=1948 module=txindex terrad_1 | 8:48PM INF Timed
out dur=4990.525864 height=1949 module=consensus round=0 step=1
terrad_1 | 8:48PM INF received proposal module=consensus
proposal={"Type":32,"block_id":{"hash":"43FEBF954F5FFEA7350E4D8D38FCCA39E439F
parts":
{"hash":"112F8B0B6AA51D1D0FCCE2D049F739EC7FEB9B6C2092B1141A55E88AC453FF57","
total":1}},"height":1 949,"pol_round":-
1,"round":0,"signature":"tuusiNhk5kRrKpLykCNWb/4b1nrfk22n-
RCyaLh650ouCugzK7rGOHq+4eQvBu3HQ3/Y4mF/RFbA67tqO2+nrBQ==","timestamp":"
2022-05-08T20:48:17.164422986Z"} terrad_1 | 8:48PM INF received
complete proposal block
hash=43FEBF954F5FFEA7350E4D8D38FCCA39E439F2D3026ADE9ED3E21D2706AC2596
height=1949 module=consensus terrad_1 | 8:48PM INF finalizing commit
of block
hash=43FEBF954F5FFEA7350E4D8D38FCCA39E439F2D3026ADE9ED3E21D2706AC2596
height=1949 module=consensus num_txs=0
root=621428F6926740D55F70759F5D9CD33E2056117DEFD33525C3434F4150FC 02B9
terrad_1 | 8:48PM INF minted coins from module account
amount=226578918uluna from=mint module=x/bank terrad_1 | 8:48PM INF
executed block height=1949 module=state num_invalid_txs=0
num_valid_txs=0 terrad_1 | 8:48PM INF commit synced
commit=436F6D6D697449447B5B352032323420323437203734203133382033312
032313220353220323432203136372031343420373820383320363920323037203 13
939203133352033372032303420323820333620313533203339203139302032323 82
033302031343420353820313834203234362031383520323037 5D3A3739447D
terrad_1 | 8:48PM INF committed state
app_hash=05E0F74A8A1FD434F2A7904E5345CFC78725CC1C249927BEE41E903AB8F6B9CF
height=1949 module=state num_txs=0 terrad_1 | 8:48PM INF indexed block
height=1949 module=txindex terrad_1 | 8:48PM INF Timed out
dur=4987.072234 height=1950 module=consensus round=0 step=1 terrad_1 |
8:48PM INF received proposal module=consensus
```

```
proposal={"Type":32,"block_id":{"hash":"5E1C9EA0F662335-
DAD4936E50C904AF1FF3D8BBF1083A967A8C4D8EA762942F7","parts":
{"hash":"0C3DA2F2B1CDCDF56C432F85E3EE1B0A21064EF69E4DB2D5CAFBE0609FDDE699","
total":1}},"height":1950,"pol_round":-
1,"round":0,"signature":"ZQeA6WQaamDpW07PD64ezc+U1vs/IW3XzMp/yShIIMbJChJFiOJw
dtipWPORdKHNrWVVFlei4drGlx70iCg==","timestamp":"2022- 05-
08T20:48:22.181798507Z"} terrad_1 | 8:48PM INF received complete
proposal block
hash=5E1C9EA0F662335DAD4936E50C904AF1FF3D8BBF1083A967A8C4D8EA762942F7
height=1950 module=consensus terrad_1 | 8:48PM INF finalizing commit
of block hash=5E1C9EA0F662335-
DAD4936E50C904AF1FF3D8BBF1083A967A8C4D8EA762942F7 height=1950
module=consensus num_txs=0
root=05E0F74A8A1FD434F2A7904E5345CFC78725CC1C249927BEE41E903AB8F6B9CF
terrad_1 | 8:48PM INF minted coins from module account
amount=226578923uluna from=mint module=x/bank terrad_1 | 8:48PM INF
executed block height=1950 module=state num_invalid_txs=0
num_valid_txs=0 terrad_1 | 8:48PM INF commit synced
commit=436F6D6D697449447B5B32313320313334203234392039332031383020
3
13234203136342033372033342032313620313733203236203734203133322032343
22032352031383520313731203230322037302032333520313635203339203235352
03235322032333020353120372031342031352035372031303 35D3A3739457D
terrad_1 | 8:48PM INF committed state app_hash=D586F95DB47-
CA42522D8AD1A4A84F219B9ABCA46EBA527FFFCE633070E0F3967 height=1950
module=state num_txs=0 terrad_1 | 8:48PM INF indexed block height=1950
module=txindex terrad_1 | 8:48PM INF Timed out dur=4983.73452
height=1951 module=consensus round=0 step=1 terrad_1 | 8:48PM INF
received proposal module=consensus
proposal={"Type":32,"block_id":{"hash":"2D43F5FF6625BD376D71F60269615A5E556BE
parts":
{"hash":"8377DFE5E66B80B3B9DA5894F5778DF1585741A1C718626AE14167158F7A5DAE","
total":1}},"height":1951,"pol_round ":-
1,"round":0,"signature":"Xtc1Wggm2dD/n3YNFjvykNUmXFkN9gNuVL/
yKab4js2DMl2xGJ7ZAjDW/0RceJIBle9SWX13smUamxfsd2wbCQ==","timestamp":"
2022-05-08T20:48:27.197935715Z"} terrad_1 | 8:48PM INF received
complete proposal block
hash=2D43F5FF6625BD376D71F60269615A5E556BEFF85FC44461B05AF17E54958490
height=1951 module=consensus terrad_1 | 8:48PM INF finalizing commit
of block
hash=2D43F5FF6625BD376D71F60269615A5E556BEFF85FC44461B05AF17E54958490
height=1951 module=consensus num_txs=1 root=D586F95DB47-
CA42522D8AD1A4A84F219B9ABCA46EBA527FFFCE633070E0F3967 terrad_1 |
8:48PM INF minted coins from module account amount=226578928uluna
from=mint module=x/bank terrad_1 | 8:48PM INF executed block
height=1951 module=state num_invalid_txs=0 num_valid_txs=1
```

Interacting with Local Terra Using Python: Storing the Contract

Then you can create a new folder inside the local Terra directory and name it testing. Then create another new folder inside the testing folder and name it artifacts. From the artifacts folder of some of the created projects in the local

Terra directory (Like `NewProjectName`), copy the `.wasm` file and paste it into the new artifacts folder. After that create a new python file inside of the testing folder and name it `contract.py`. You can write the following code in the `contract.py`.

```
from terra_sdk.client.localterra import LocalTerra
from terra_sdk.util.contract import read_file_as_b64
from terra_sdk.core.fee import Fee
from terra_sdk.client.lcd.api.tx import CreateTxOptions
from terra_sdk.core.wasm import MsgStoreCode

lt = LocalTerra()
deployer = lt.wallets["test1"]

def store_contract(contract_name:str) -> str:
 contract_bytes = read_file_as_b64(f"artifacts/{contract_name}.wasm")
 store_code = MsgStoreCode(
 deployer.key.acc_address,
 contract_bytes
 )
 tx = deployer.create_and_sign_tx(CreateTxOptions(
  msgs = [store_code],
  fee = Fee(4000000,"10000000uluna")
  ))
 result = lt.tx.broadcast(tx)
print(result)

store_contract("Testing")
```

In the above code, at first, we connect to local Terra, then we create a wallet for the deployer of the contract (which is us). After that, we create a function called store contract, inside which we read the `.wasm` file in the artifacts folder and store it in a variable called `contract_bytes`. Then we create a message called the `MsgStoreCode` using the `contract_bytes` variable and after that, we create and sign a transaction using the `store_code message`. Finally, we broadcast the transaction and use the `store_contract` function with the name of the contract called Testing (The name of our project folder).

And run it in another bash, so as not to interfere with the docker running the local Terra, use the following command to start our first interaction with Terra network. `python3 contract.py` Result:

```
BlockTxBroadcastResult(height=1626,
txhash='1143BD1F71A9E951F3B99D7EEE4BF0EE304F0C6C4A63DADA96F7CE243F25F39A',
raw_log='[{"events":[{"type":"message","attributes":
[{"key":"action","value":"/terra.wasm.v1beta1.MsgStoreCode"},
{"key":"module","value":"wasm"}]},{"type":"store_code","attributes":
[{"key":"sender","value":"terra1x46rqay4d3cssq8gxxvqz8xt6nwlz4td20k38v"},
{"key":"code_id","value":"1"}]}]}]', gas_wanted=4000000,
```

```
gas_used=1606062, logs=[TxLog(msg_index=0, log='', events=[{'type':
'message', 'attributes': [{'key': 'action', 'value':
'/terra.wasm.v1beta1.MsgStoreCode'}, {'key': 'module', 'value':
'wasm'}]}, {'type': 'store_code', 'attributes': [{'key': 'sender',
'value': 'terra1x46rqay4d3cssq8gxxvqz8xt6nwlz4td20k38v'}, {'key':
'code_id', 'value': '1'}]}], events_by_type={'message': {'action':
['/terra.wasm.v1beta1.MsgStoreCode'], 'module': ['wasm']},
'store_code': {'sender':
['terra1x46rqay4d3cssq8gxxvqz8xt6nwlz4td20k38v'], 'code_id':
['1']}})], code=0, codespace='', info=None, data=None, timestamp=
None)
```

In the above dictionary, you can see data like `txhash` (transaction hash), `code_id`, sender of the `store_code`, and so on.

Interacting with Local Terra Using Python: Getting the Code ID

Now, let's get the code id of the transaction using the very same code and making some small changes:

```
from terra_sdk.client.localterra import LocalTerra
from terra_sdk.util.contract import read_file_as_b64,get_code_id
from terra_sdk.core.fee import Fee
from terra_sdk.client.lcd.api.tx import CreateTxOptions
from terra_sdk.core.wasm import MsgStoreCode

lt = LocalTerra()
deployer = lt.wallets["test1"]

def store_contract(contract_name:str) -> str:
 contract_bytes = read_file_as_b64(f"artifacts/{contract_name}.wasm")
 store_code = MsgStoreCode(
 deployer.key.acc_address,
 contract_bytes
 )
 tx = deployer.create_and_sign_tx(CreateTxOptions(
  msgs = [store_code],
  fee = Fee(4000000,"10000000uluna")
  ))
 result = lt.tx.broadcast(tx)
 code_id = get_code_id(result)
return(code_id)

print(store_contract("Testing"))
```

We have written the very same code as the previous one with the difference that this time we have specifically returned the `code_id` in the dictionary and have printed it. Now, let's run the code using the following command on the terminal, and remember not to lose your connection with local Terra (keep the docker up and running).

`python3 contract.py` Result: `3` Let us try another time: `python3 contract.py`

Result: `4` And one more time to make sure everything works as it should: `python3 contract.py` Result: `5`

Sending JSON Messages to Local Terra Smart Contracts Using Python

The following article is the continuation of the previous one and here we try to interact with local Terra smart contracts using python scripts and at the same time using the CosmWasm functions inside `LocalTerra\src` folder written in Rust programming language. We will instantiate from the contract and execute it by sending JSON messages to Local Terra smart contracts.

Instantiating form the Contract of the Local Terra

In the following code, we use the `MsgInstantiateContract` from `terra_sdk.core.wasm` to instantiate from the contract, using the `code_id` and return the contract address:

```
from terra_sdk.client.localterra import LocalTerra
from terra_sdk.util.contract import
 read_file_as_b64,get_code_id, get_contract_address
from terra_sdk.core.fee import Fee
from terra_sdk.client.lcd.api.tx import CreateTxOptions
from terra_sdk.core.wasm import MsgStoreCode,MsgInstantiateContract

lt = LocalTerra()
deployer = lt.wallets["test1"]

def store_contract(contract_name:str) -> str:
contract_bytes = read_file_as_b64(f"artifacts/{contract_name}.wasm")
store_code = MsgStoreCode(
  deployer.key.acc_address,
  contract_bytes
  )
tx = deployer.create_and_sign_tx(CreateTxOptions(
msgs = [store_code],
fee = Fee(4000000,"10000000uluna")
  ))
result = lt.tx.broadcast(tx)
code_id = get_code_id(result)
 return(code_id)

def instantiate_contract(code_id: str,init_msg)-> str:
instantiate = MsgInstantiateContract(
  deployer.key.acc_address,
  deployer.key.acc_address,
code_id = code_id,
init_msg = init_msg,
  )
tx = deployer.create_and_sign_tx(CreateTxOptions(
```

```
msgs = [instantiate],
fee = Fee(4000000,"10000000uluna")
  ))
result = lt.tx.broadcast(tx)
contract_address = get_contract_address(result)
 return contract_address

code_id = store_contract("Testing")
contract_address = instantiate_contract(code_id,{"count": 15})
print(code_id, contract_address)
```

The goal of the above code is to instantiate from the contract using the `code_id` of the `store_contract` transaction. To do so, we define a function called instantiate_contract. Inside this function, we create a message with `MsgInstantiateContract` function. Then we will create and sign the transaction using the created message. Finally, we will broadcast the transaction and return the contract address. Notice that the function that we have defined has 2 attributes: `code_id` and `init_msg`. When we call this function, we can enter the `code_id` we have got from the output of `store_contract`. For `init_msg`, we enter the `{"count": number}` in which number can be any unsigned integer number. In the end, we print the `code_id` with the contract address which is the output of the `instantiate_contract` function.

Now, let's try our script and see if it prints out the code_id and the contract address: `python3 contract.py` Result: `11 terra1wkgucw0zply6e5c4h30a4z5qljhazepw4jpf2s` Try another time: `python3 contract.py` Result: `12 terra1mhf9cr9f70rd052rptnevcje2puq6heekrxdwa` As you can see the `code_id` has incremented.

And another time: `python3 contract.py` Result: `13 terra1uw258kwqftsywjfv972y4d3f4mn0hrlkpt7y3y`

Executing the Contract

Now, using the `MsgExecuteContract`, contract address, and account address, we execute the contract inside the local Terra directory.

```
from terra_sdk.client.localterra import LocalTerra
from terra_sdk.util.contract import
 read_file_as_b64,get_code_id, get_contract_address
from terra_sdk.core.fee import Fee
from terra_sdk.client.lcd.api.tx import CreateTxOptions
from terra_sdk.core.wasm import
 MsgStoreCode,MsgInstantiateContract, MsgExecuteContract
```

```
lt = LocalTerra()
deployer = lt.wallets["test1"]

def store_contract(contract_name:str) -> str:
contract_bytes = read_file_as_b64(f"artifacts/{contract_name}.wasm")
store_code = MsgStoreCode(
 deployer.key.acc_address,
 contract_bytes
 )
tx = deployer.create_and_sign_tx(CreateTxOptions(
msgs = [store_code],
fee = Fee(4000000,"10000000uluna")
  ))
result = lt.tx.broadcast(tx)
code_id = get_code_id(result)
 return(code_id)

def instantiate_contract(code_id: str,init_msg)-> str:
instantiate = MsgInstantiateContract(
  deployer.key.acc_address,
  deployer.key.acc_address,
code_id = code_id,
init_msg = init_msg,
  )
tx = deployer.create_and_sign_tx(CreateTxOptions(
msgs = [instantiate],
fee = Fee(4000000,"10000000uluna")
  ))
result = lt.tx.broadcast(tx)
contract_address = get_contract_address(result)
 return contract_address

def execute_contract(sender,contract_addr: str, execute_msg):
execute = MsgExecuteContract(
sender = sender.key.acc_address, contract = contract_addr,execute_msg
 = execute_msg)
tx = sender.create_and_sign_tx(CreateTxOptions(
msgs = [execute],
fee = Fee(4000000,"10000000uluna")
  ))
 )
result = lt.tx.broadcast(tx)
 return result

code_id = store_contract("Testing")
contract_address = instantiate_contract(code_id,{"count": 15})
execute = execute_contract(deployer,contract_address,{"increment":{}})
execute1 = execute_contract(deployer,contract_address,{"increment"
:{}})
execute2 = execute_contract(deployer,contract_address,{"increment"
```

```
:{}})
execute3 = execute_contract(deployer,contract_address,{"increment"
:{}})
execute4 = execute_contract(deployer,contract_address,{"increment"
:{}})

print(execute1)
print(lt.wasm.contract_query(contract_address,{"get_count":{}}))
print(code_id, contract_address)
```

In the above Python script, we will use the execute_contract next to `store_contract` and `instantiate_contract` functions. In the execute function, we have done the 3 main jobs we did for other functions, creating the message, creating and signing the transaction , and broadcasting it. The execute function has 3 attributes:

1. 1. The sender (deployer of the contract).
2. 2. The contract address that we have got from instantiate_contract function.
3. 3. The execute message which can be increment, reset ,etc.

Outside the execute function, we call it 5 times with the increment message. And to see the results, we print the second execution and also the query result of the `get_count` message which is going to check if we have successfully executed the contract 5 times. If so, the result must show that the `count = 20. (15 +1+1+1+1+1 = 20)`.

Now, let's see if our code prints out the result of local Terra contract execution, as well as the `code_id`, query result, and the contract address: `python3 contract.py` Result:

```
BlockTxBroadcastResult(height=8030,
txhash='F51A179323A3489F4629FB0ECA4B83C64237961F9244409EFBDBF88EFC8CFA97',
raw_log='[{"events":
[{"type":"execute_contract","attributes":[{"key":"sender","value":"terra1x46r
{"key":"contract_address","value":"terra1hteg5pjt0r27589yd67jwhq8an4j3w4u02qf
type":"from_contract","attributes":[{"key":"contract_
address","value":"terra1hteg5pjt0r27589yd67jwhq8an4j3w4u02qfmz"},
{"key":"method","value":"try_increment"}]},{"type":"message","attributes":
[{"key":"action","value":"/terra.wasm.v1beta1.MsgExecuteContract"},{"key":"mo
value":"wasm"},{"key":"sender","value":"terra1x46rqay4d3cssq8gxxvqz8xt6nwlz4t
type":"wasm","attributes":
[{"key":"contract_address","value":"terra1hteg5pjt0r27589yd67jwhq8an4j3w4u02c
key":"method","value":"try_increment"}]}]}]', gas_wanted=4000000,
gas_used=102492, logs=[TxLog(msg_index=0, log='', events=[{'type':
'execute_contract', 'attributes': [{'key': 'sender', 'value':
'terra1x46rqay4d3cssq8gxxvqz8xt6nwlz4td20k38v'}, {'key':
'contract_address', 'value':
'terra1hteg5pjt0r27589yd67jwhq8an4j3w4u02qfmz'}]}, {'type':
'from_contract', 'attributes': [{'key': 'contract_address', 'value':
```

'terra1hteg5pjt0r27589yd67jwhq8an4j3w4u02qfmz'}, {'key': 'method', 'value': 'try_increment'}]}, {'type': 'message', 'attributes': [{'key': 'action', 'value': '/terra.wasm.v1beta1.MsgExecuteContract'}, {'key': 'module', 'value': 'wasm'}, {'key': 'sender', 'value': 'terra1x46rqay4d3cssq8gxxvqz8xt6nwlz4td20k38v'}]}, {'type': 'wasm', 'attributes': [{'key': 'contract_address', 'value': 'terra1hteg5pjt0r27589yd67jwhq8an4j3w4u02qfmz'}, {'key': 'method', 'value': 'try_increment'}]}], events_by_type={'execute_contract': {'sender': ['terra1x46rqay4d3cssq8gxxvqz8xt6nwlz4td20k38v'], 'contract_address': ['terra1hteg5pjt0r27589yd67jwhq8an4j3w4u02qfmz']}, 'from_contract': {'contract_address': ['terra1hteg5pjt0r27589yd67jwhq8an4j3w4u02qfmz'], 'method': ['try_increment']}, 'message': {'action': ['/terra.wasm.v1beta1.MsgExecuteContract'], 'module': ['wasm'], 'sender': ['terra1x46rqay4d3cssq8gxxvqz8xt6nwlz4td20k38v']}, 'wasm': {'contract_address': ['terra1hteg5pjt0r27589yd67jwhq8an4j3w4u02qfmz'], 'method': ['try_increment']}})], code=0, codespace='', info=None, data=None, timestamp=None) {'count': 20} 17 terra1hteg5pjt0r27589yd67jwhq8an4j3w4u02qfmz Notice that it takes a few seconds for the contract to be instantiated and the transaction to be executed. As we have determined the `count = 15` in the `instantiate_contract` function and executed the contract with the increment message 5 times, The result of the query count will be `20`.

Try one more time, and you will see the same result with the `code_id` incremented. `python3 contract.py` Result:

`BlockTxBroadcastResult(height=8269, txhash='73C112C7B0DA968209310068DFCA9C26378A47EA383C3A4DBE350DCFE54079E8', raw_log='[{"events": [{"type":"execute_contract","attributes":[{"key":"sender","value":"terra1x46r {"key":"contract_address","value":"terra1wfvvdh7vtq82xz3x6h63ul9xn8hxh5xhkwl5 type":"from_contract","attributes": [{"key":"contract_address","value":"terra1wfvvdh7vtq82xz3x6h63ul9xn8hxh5xhkwl key":"method","value":"try_increment"}]},{"type":"message","attributes": [{"key":"action","value":"/terra.wasm.v1beta1.MsgExecuteContract"},{"key":"mo value":"wasm"},{"key":"sender","value":"terra1x46rqay4d3cssq8gxxvqz8xt6nwlz4t type":"wasm","attributes": [{"key":"contract_address","value":"terra1wfvvdh7vtq82xz3x6h63ul9xn8hxh5xhkwl key":"method","value":"try_increment"}]}]}]', gas_wanted=4000000, gas_used=102492, logs=[TxLog(msg_index=0, log='', events=[{'type': 'execute_contract', 'attributes': [{'key': 'sender', 'value': 'terra1x46rqay4d3cssq8gxxvqz8xt6nwlz4td20k38v'}, {'key': 'contract_address', 'value': 'terra1wfvvdh7vtq82xz3x6h63ul9xn8hxh5xhkwl5ra'}]}, {'type': 'from_contract', 'attributes': [{'key': 'contract_address', 'value': 'terra1wfvvdh7vtq82xz3x6h63ul9xn8hxh5xhkwl5ra'}, {'key': 'method', 'value': 'try_increment'}]}, {'type': 'message', 'attributes': [{'key': 'action', 'value': '/terra.wasm.v1beta1.MsgExecuteContract'}, {'key': 'module', 'value': 'wasm'}, {'key': 'sender', 'value': 'terra1x46rqay4d3cssq8gxxvqz8xt6nwlz4td20k38v'}]}, {'type': 'wasm', 'attributes': [{'key': 'contract_address', 'value':`

```
'terra1wfvvdh7vtq82xz3x6h63ul9xn8hxh5xhkwl5ra'}, {'key': 'method',
'value': 'try_increment'}]}], events_by_type={'execute_contract':
{'sender': ['terra1x46rqay4d3cssq8gxxvqz8xt6nwlz4td20k38v'],
'contract_address': ['terra1wfvvdh7vtq82xz3x6h63ul9xn8hxh5xhkwl5ra']},
'from_contract': {'contract_address':
['terra1wfvvdh7vtq82xz3x6h63ul9xn8hxh5xhkwl5ra'], 'method':
['try_increment']}, 'message': {'action':
['/terra.wasm.v1beta1.MsgExecuteContract'], 'module': ['wasm'],
'sender': ['terra1x46rqay4d3cssq8gxxvqz8xt6nwlz4td20k38v']}, 'wasm':
{'contract_address': ['terra1wfvvdh7vtq82xz3x6h63ul9xn8hxh5xhkwl5ra'],
'method': ['try_increment']}})], code=0, codespace='', info=None,
data=None, timestamp=None) {'count': 20} 18
terra1wfvvdh7vtq82xz3x6h63ul9xn8hxh5xhkwl5ra
```

Wrapping Up

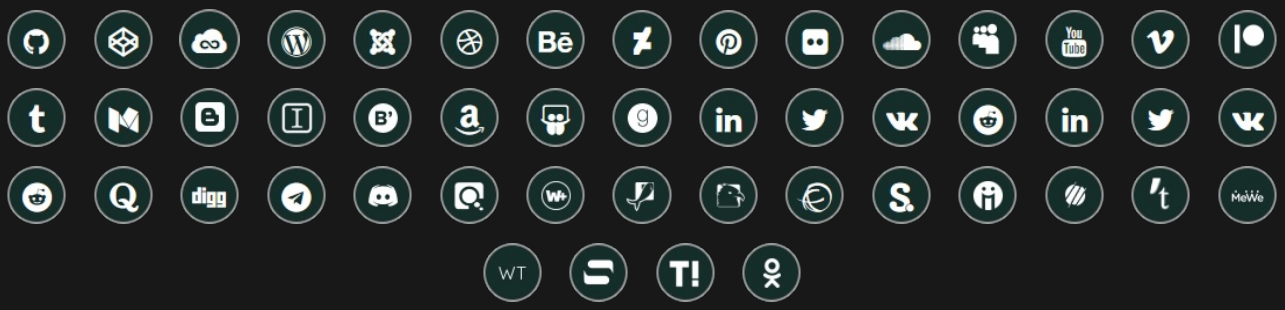
In this article, we have managed to connect to local Terra using Docker and interact with it by the means of Python scripts. In detail, we have managed to store the contract and get the `code_id` of the transaction. In the next parts, we will focus on the other functions such as instantiating and executing the contract to complete the process of interacting with Terra smart contracts.

Finally, we have managed to interact with local Terra using the Python scripts. At first, we connected to local Terra using Docker. Then, we created a wallet, that, stored our contract, instantiated from the stored contract using the `code_id`, and executed it.

Join Arashtad Community

Follow Arashtad on Social Media

We provide variety of content, products, services, tools, tutorials, etc. Each social profile according to its features and purpose can cover only one or few parts of our updates. We can not upload our videos on SoundCloud or provide our eBooks on Youtube. So, for not missing any high quality original content that we provide on various social networks, make sure you follow us on as many social networks as you're active in. You can find out Arashtad's profiles on different social media services.



Get Even Closer!

Did you know that only one universal Arashtad account makes you able to log into all Arashtad network at once? Creating an Arashtad account is free. Why not to try it? Also, we have regular updates on our newsletter and feed entries. Use all these benefitial free features to get more involved with the community and enjoy the many products, services, tools, tutorials, etc. that we provide frequently.

SIGN UP | NEWSLETTER | RSS FEED